ĐẠI HỌC ĐÀ NẴNG

TRƯỜNG CAO ĐẲNG CÔNG NGHỆ THÔNG TIN

GI ÁO TRÌ NH

LẬP TRÌ NH ỨNG DỤNG MẠNG

Người soạn: Mai Lam

Đà Nẵng, 2010

MỤC LỤC

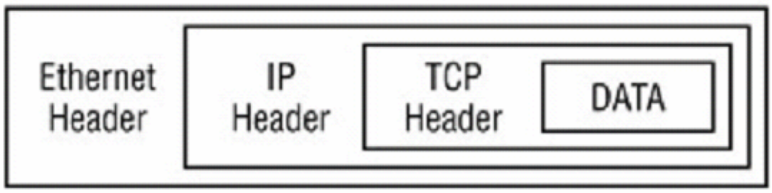
CHƯƠNG I. NHỮNG KIẾN THỨC CƠ BẢN VỀ LẬP TRÌNH MẠNG

I.1. Tổng quan

Internet Protocol (IP) là nền tảng của lập trình mạng. IP là phương tiện truyền tải dữ liệu giữa các hệ thống bất kể đó là hệ thống mạng cục bộ (LAN) hay hệ thống mạng diện rộng (WAN). Mặc dù lập trình viên mạng có thể chọn các giao thức khác để lập trình nhưng IP cung cấp các kỹ thuật mạnh nhất để gởi dữ liệu giữa các thiết bị, đặc biệt là thông qua mạng Internet.

Để hiểu rõ các khái niệm bên dưới lập trình mạng, chúng ta phải hiểu rõ giao thức IP, hiểu cách nó chuyển dữ liệu giữa các thiết bị mạng. Lập trình mạng dùng giao thức IP thường rất phức tạp. Có nhiều yếu tố cần quan tâm liên quan đến cách dữ liệu được gởi qua mạng: số lượng Client và Server, kiểu mạng, tắc nghẽn mạng, lỗi mạng,… Bởi vì các yếu tố này ảnh hưởng đến việc truyền dữ liệu từ thiết bị này đến thiết bị khác trên mạng do đó việc hiểu rõ chúng là vấn đề rất quan trọng để lập trình mạng được thành công.

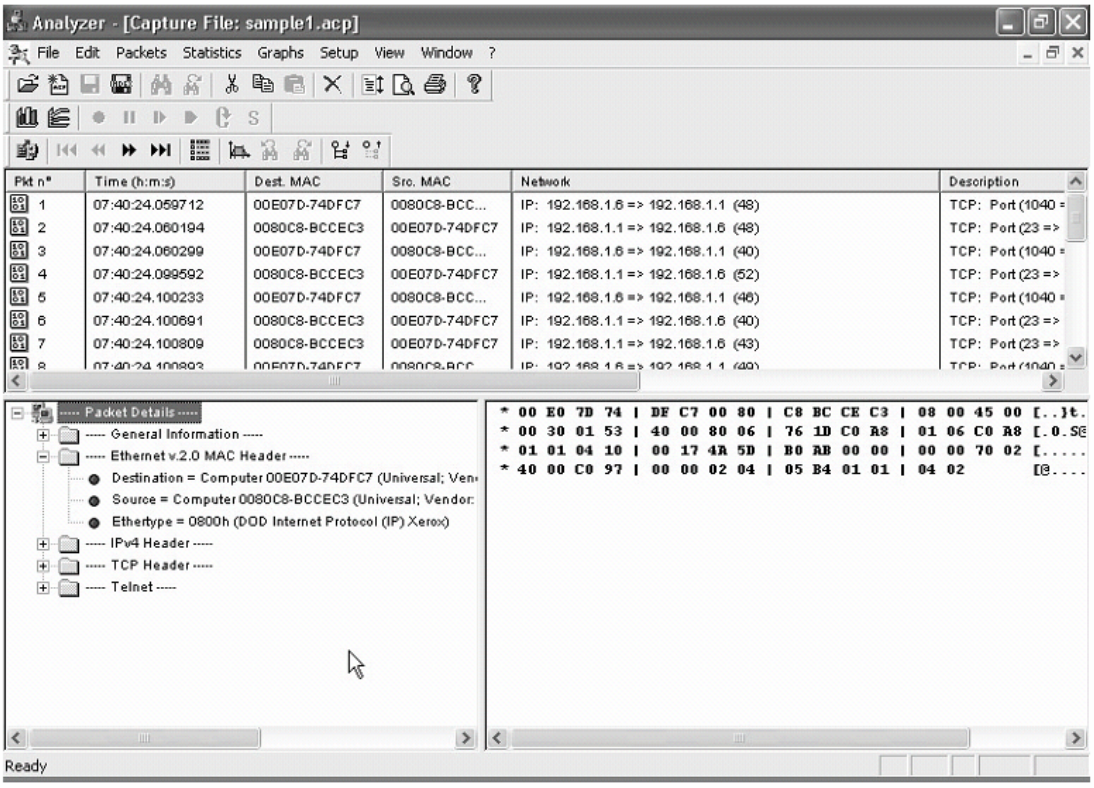
Một gói dữ liệu mạng gồm nhiều tầng thông tin. Mỗi tầng thông tin chứa một dãy các byte được sắp đặt theo một trật tự đã được định sẵn. Hầu hết các gói dữ liệu dùng trong lập trình mạng đều chứa ba tầng thông tin cùng với dữ liệu được dùng để truyền tải giữa các thiết bị mạng. Hình sau mô tả hệ thống thứ bậc của một gói IP:



Hình I.1: Các tầng giao thức mạng trong các gói dữ liệu

I.1.1. Tầng Ethernet

Tầng đầu tiên của gói dữ liệu mạng được gọi là Ethernet Header, trong tầng này có ba gói giao thức Ethernet: Ethernet 802.2, Ethernet 802.3, và Ethernet phiên bản 2. Các giao thức Ethernet 802.2 và Ethernet 802.3 là các giao thức chuẩn của IEEE. Ethernet phiên bản 2 tuy không phải là giao thức chuẩn nhưng nó được sử dụng rộng rãi trong mạng Ethernet. Hầu hết các thiết bị mạng kể cả hệ điều hành Windows mặc định dùng giao thức Ethernet phiên bản 2 để truyền tải các gói IP.



Hình I.2: Ethernet Header

Phần đầu của Ethernet phiên bản 2 là địa chỉ MAC (Media Access Card) dùng để xác định các thiết bị trên mạng cùng với số giao thức Ethernet xác định giao thức tầng tiếp theo chứa trong gói Ethernet. Mỗi gói Ethernet bao gồm:

- 6 byte địa chỉ MAC đích
- 6 byte địa chỉ MAC nguồn
- 2 byte xác định giao thức tầng kế tiếp
- Data payload từ 46 đến 1500 byte
- 4-byte checksum

I.1.2. Địa chỉ Ethernet

Địa chỉ Ethernet (địa chỉ MAC) là địa chỉ của các thiết bị, địa chỉ này được gán bởi các nhà sản xuất thiết bị mạng và nó không thay đổi được. Mỗi thiết bị trên mạng Ethernet phải có 1 địa chỉ MAC duy nhất. Địa chỉ MAC gồm 2 phần:

- 3 byte xác định nhà sản xuất
- 3 byte xác định số serial duy nhất của nhà sản xuất

Giản đồ địa chỉ Ethernet cho phép gán các địa chỉ broadcast hoặc multicast. Đối với địa chỉ broadcast thì tất cả các bit của địa chỉ đích được gán bằng 1 (FFFFFFFFFFFF). Mỗi thiết bị mạng sẽ chấp nhận các gói có địa chỉ broadcast. Địa chỉ này hữu ích cho các giao thức phải gởi các gói truy vấn đến tất cả các thiết bị mạng. Địa chỉ multicast cũng là một loại địa chỉ đặc biệt của địa chỉ Ethernet, các địa chỉ multicast chỉ cho phép một số các thiết bị chấp nhận gói tin. Một số địa chỉ Ethernet multicast:

Địa chỉ	Mô tả
01-80-C2-00-00-00	Spanning tree (for bridges)
09-00-09-00-00-01	HP Probe
09-00-09-00-00-04	HP DTC
09-00-2B-00-00-00	DEC MUMPS
09-00-2B-00-00-01	DEC DSM/DTP
09-00-2B-00-00-02	DEC VAXELN
09-00-2B-00-00-03	DEC Lanbridge Traffic Monitor (LTM)
09-00-2B-00-00-04	DEC MAP End System Hello
09-00-2B-00-00-05	DEC MAP Intermediate System Hello
09-00-2B-00-00-06	DEC CSMA/CD Encryption
09-00-2B-00-00-07	DEC NetBios Emulator
09-00-2B-00-00-0F	DEC Local Area Transport (LAT)
09-00-2B-00-00-1x	DEC Experimental
09-00-2B-01-00-00	DEC LanBridge Copy packets (all bridges)
09-00-2B-02-00-00	DEC DNA Lev. 2 Routing Layer Routers
09-00-2B-02-01-00	DEC DNA Naming Service Advertisement
09-00-2B-02-01-01	DEC DNA Naming Service Solicitation
09-00-2B-02-01-02	DEC DNA Time Service
09-00-2B-03-xx-xx	DEC default filtering by bridges
09-00-2B-04-00-00	DEC Local Area System Transport (LAST)
09-00-2B-23-00-00	DEC Argonaut Console
09-00-4E-00-00-02	Novell IPX
09-00-77-00-00-01	Retix spanning tree bridges
09-00-7C-02-00-05	Vitalink diagnostics
09-00-7C-05-00-01	Vitalink gateway
0D-1E-15-BA-DD-06	HP
CF-00-00-00-00-00	Ethernet Configuration Test protocol (Loopback)

I.1.3. Ethernet Protocol Type

Một phần khác rất quan trọng của Ethernet Header là trường Protocol Type, trường này có kích thước hai byte. Sự khác nhau giữa gói tin Ethernet phiên bản 2 và Ethernet 802.2 và 802.3 xảy ra ở trường này. Các gói tin Ethernet 802.2 và 802.3 sử dụng trường này để cho biết kích thước của một gói tin Ethernet. Ethernet phiên bản 2 dùng trường này để định nghĩa giao thức tầng kế tiếp trong gói tin Ethernet. Một số giá trị của trường này:

Giá trị	Giao thức
0800	IP
0806	ARP
0800	IP
0BAD	Banyan VINES
8005	HP Probe
8035	Reverse ARP
809B	AppleTalk
80D5	IBM SNA
8137	Novell
8138	Novell
814C	Raw SNMP
86DD	IPv6
876B	TCP/IP compression

I.1.4. Data payload

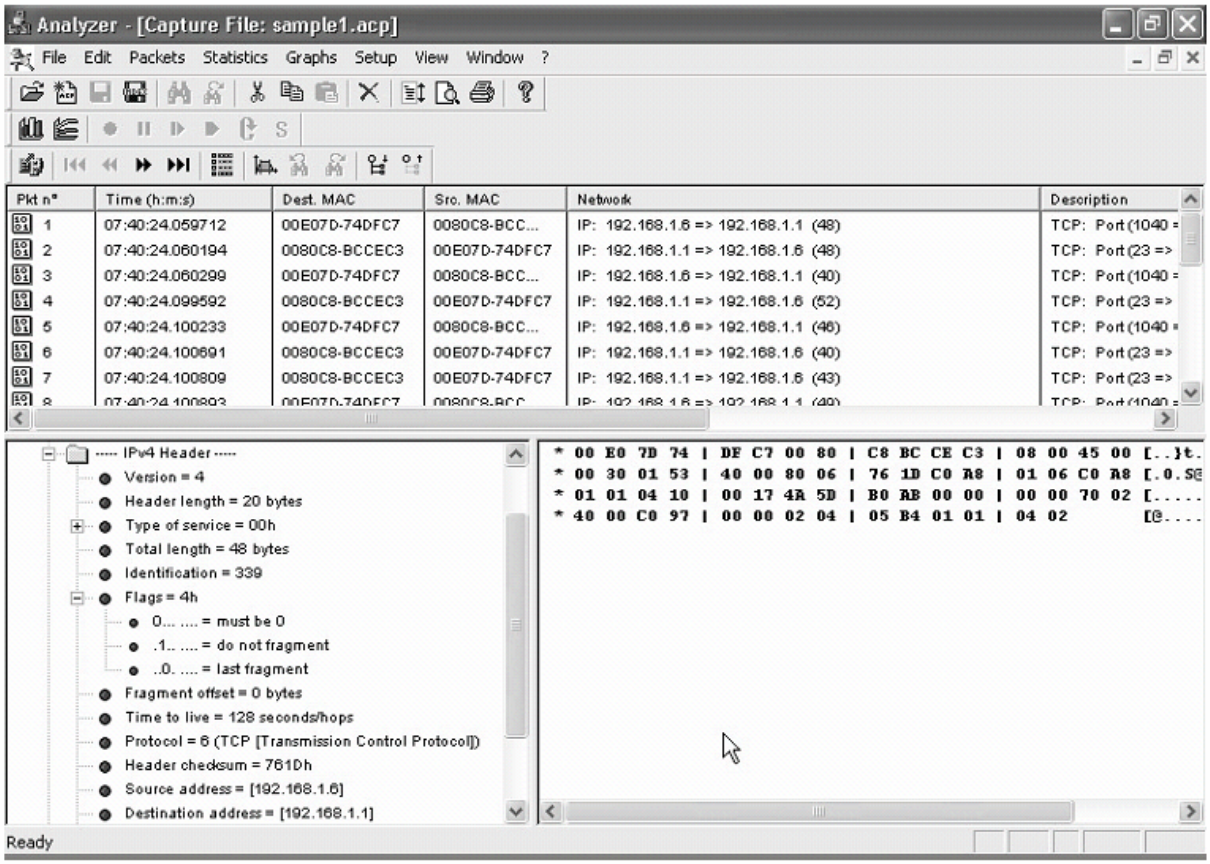
Data payload phải chứa tối thiểu 46 byte để đảm bảo gói Ethernet có chiều dài tối thiểu 64 byte. Nếu phần data chưa đủ 46 byte thì các ký tự đệm được thêm vào cho đủ. Kích thước của trường này từ 46 đến 1500 byte.

I.1.5. Checksum

Giá trị checksum cung cấp cơ chế kiểm tra lỗi cho dữ liệu, kích thước của trường này là 4 byte . Nếu gói tin bị hỏng trong lúc truyền, giá trị checksum sẽ bị tính toán sai và gói tin đó được đánh dấu là gói tin xấu.

I.2. Tầng IP

Tầng IP định nghĩa thêm nhiều trường thông tin của của giao thức Ethernet



Hình I.3: Thông tin tầng IP

Các trường trong tầng IP:

Trường	**Bit**	**Mô tả**
Version	4	Phiên bản IP header (phiên bản hiện tại là 4)
Header Length	4	Chiều dài phần header của gói IP
Type of Service	8	Kiểu chất lượng dịch vụ QoS (Quality of Service)
Total Length	16	Chiều dài của gói IP
Identification	16	Giá trị ID duy nhất xác định các gói IP
Flags	3	Cho biết gói IP có bị phân đoạn hay không hay còn các phân đoạn khác
Fragment offset	13	Vị trí của phân đoạn trong gói IP
Time to Live (TTL)	8	Thời gian tối đa gói tin được phép ở lại trên mạng (được tính bằng giây)
Protocol	8	Kiểu giao thức của tầng dữ liệu kế tiếp
Header Checksum	16	Checksum của dữ liệu gói IP header
Source Address	32	Địa chỉ IP của thiết bị gởi
Destination Address	32	Địa chỉ IP của thiết bị nhận
Options		Định nghĩa các đặc điểm của gói IP trong tương lai

I.2.1. Trường địa chỉ

Địa chỉ Ethernet dùng để xác định các thiết bị trên mạng LAN nhưng nó không thể dùng để xác định địa chỉ của các thiết bị trên mạng ở xa. Để xác định các thiết bị trên các mạng khác nhau, địa chỉ IP được dùng. Một địa chỉ IP là một số 32 bit và địa chỉ IP được chia thành 4 lớp sau:

Lớp A	0.x.x.x–127.x.x.x
Lớp B	128.x.x.x–191.x.x.x
Lớp C	192.x.x.x–223.x.x.x
Lớp D	224.x.x.x–254.x.x.x

I.2.2. Các cờ phân đoạn

Một trong những phức tạp, rắc rối của gói IP là kích thước của chúng. Kích thước tối đa của gói IP có thể lên đến 65,536 byte. Đây là một lượng rất lớn dữ liệu cho một gói tin. Thực tế hầu hết các truyền tải dữ liệu ở cấp thấp như Ethernet không thể hỗ trợ một gói IP lớn (phần dữ liệu của Ethernet chỉ có thể tối đa 1500 byte). Để giải quyết vấn đề này, các gói IP dùng fragmentation (phân đoạn) để chia các gói IP thành các phần nhỏ hơn để truyền tải tới đích. Khi các mảnh được truyền tải tới đích, phần mềm của thiết bị nhận phải có cách để nhận ra các phân đoạn của gói tin và ráp chúng lại thành thành 1 gói IP.

Sự phân đoạn được thành lập nhờ vào việc sử dụng 3 trường của gói IP: fragmentation flags, fragment offset, và trường identification. Mỗi cờ phân đoạn bao gồm ba cờ một bit sau:

• Cờ reserved: giá trị zero

• Cờ Don’t Fragment: cho biết gói IP không bị phân đoạn

• Cờ More Fragment: cho biết gói tin bị phân đoạn và còn các phân đoạn khác nữa

Trường IP Identification xác định duy nhất định danh mỗi gói IP. Tất cả các phân đoạn của bất kỳ gói IP nào cũng đều có cùng số identification. Số identification giúp cho phần mềm máy nhận biết được các phân đoạn nào thuộc gói IP nào và ráp lại cho đúng.

Trường Fragment offset cho biết vị trí của phân đoạn trong gói tin ban đầu.

I.2.3. Trường Type of Service

Trường Type of Service xác định kiểu chất lượng dịch vụ QoS (Quality of Service) cho gói IP. Trường này được dùng để đánh dấu một gói IP có một độ ưu tiên nào đó chẳng hạn như được dùng để tăng độ ưu tiên của các dữ liệu cần thời gian thực như Video, Audio.

Trong hầu hết các truyền tải mạng, trường này được được thiết lập giá trị zero, cho biết đây là dữ liệu bình thường, tuy nhiên với các ứng dụng cần thời gian thực như Video hay Audio thì trường này sẽ được sử dụng để tăng độ ưu tiên cho gói dữ liệu.

Trường này gồm tám bit và ý nghĩa các bit như sau:

• 3 bit được dùng làm trường ưu tiên

• 1 bit cho biết thời gian trễ là bình thường hay thấp

• 1 bit cho biết thông lượng bình thường hay cao

• 1 bit cho biết độ tin cậy bình thường hay cao

• 2 bit được dùng trong tương lai

I.2.4. Trường Protocol

Được dùng để xác định giao thức tầng tiếp theo trong gói IP, IANA định nghĩa 135 giá trị cho trường này có thể dùng trong gói IP nhưng chỉ có một số giá trị hay được dùng trong bảng sau:

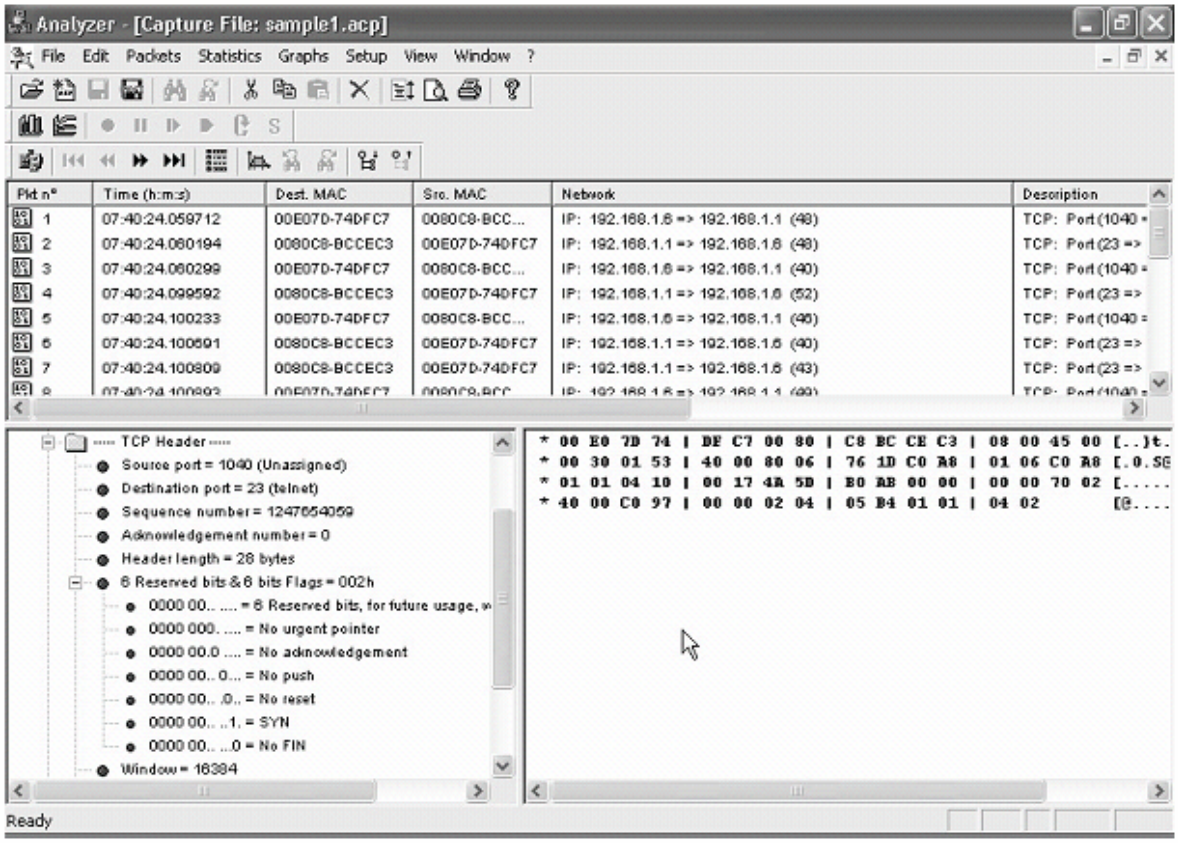
Giá trị	**Giao thức**
1	Internet Control Message (ICMP)
2	Internet Group Message (IGP)
6	Transmission Control (TCP)
8	Exterior Gateway (EGP)
9	Interior Gateway (Cisco IGP)
17	User Datagram (UDP)
88	Cisco EIGRP

Hai giao thức được dùng nhiều nhất trong lập trình mạng là TCP và UDP

I.3. Tầng TCP

Giao thức TCP (Transmission Control Protocol) là giao thức hướng kết nối, nó cho phép tạo ra kết nối điểm tới điểm giữa hai thiết bị mạng, thiết lập một đường nhất quán để truyền tải dữ liệu. TCP đảm bảo dữ liệu sẽ được chuyển tới thiết bị đích, nếu dữ liệu không tới được thiết bị đích thì thiết bị gởi sẽ nhận được thông báo lỗi.

Các nhà lập trình mạng phải hiểu cách hoạt động cơ bản của TCP và đặc biệt là phải hiểu cách TCP truyền tải dữ liệu giữ các thiết bị mạng. Hình sau cho thấy những trường của TCP Header. Những trường này chứa các thông tin cần thiết cho việc thực thi kết nối và truyền tải dữ liệu một cách tin tưởng.



Hình I.4: Các trường của TCP Header

Mỗi trường của TCP Header kết hợp với một chức năng đặc biệt của một phiên làm việc TCP. Có một số chức năng quan trọng sau:

• Source port và Destination port: theo dõi các kết nối giữa các thiết bị

• Sequence và Acknowledgement number: theo dõi thứ tự các gói tin và truyền tải lại các gói tin bị mất

• Flag: mở và đóng kết nối giữa các thiết bị để truyền tải dữ liệu

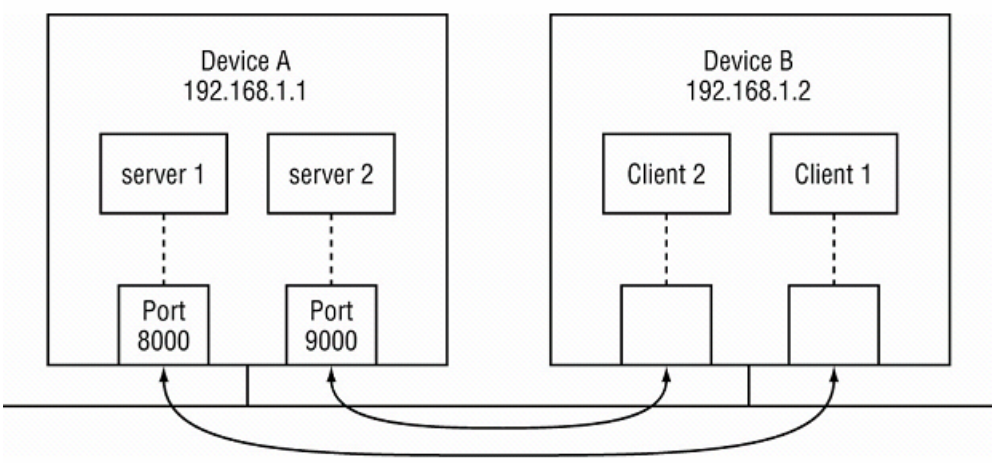
I.3.1. TCP port

TCP sử dụng các port để xác định các kết nối TCP trên một thiết bị mạng. Để liên lạc với một ứng dụng chạy trên một thiết bị mạng ở xa ta cần phải biết hai thông tin:

• Địa chỉ IP của thiết bị ở xa

• TCP port được gán cho thiết bị ở xa

Để kết nối TCP được thành lập, thiết bị ở xa phải chấp nhận các gói tin truyền đến port đã được gán. Bởi vì có nhiều ứng dụng chạy trên một thiết bị sử dụng TCP do đó thiết bị phải cấp phát các cổng khác nhau cho các ứng dụng khác nhau.



Hình I.5: Kết nối TCP đơn giản

Trong hình trên thì thiết bị A đang chạy hai ứng dụng Server, hai ứng dụng này đang chờ các gói tin từ Client. Một ứng dụng được gán port 8000 và một ứng dụng được gán port 9000. Thiết bị mạng B muốn kết nối đến thiết bị mạng A thì nó phải được gán một TCP port còn trống từ hệ điều hành và port này sẽ được mở trong suốt phiên làm việc. Các port ở Client thường không quan trọng và có thể gán bất kỳ một port nào hợp lệ trên thiết bị.

Tổ hợp của một địa chỉ IP và một port là một IP endpoint. Một phiên làm việc TCP được định nghĩa là một sự kết hợp của một IP endpoint cục bộ và một IP endpoint ở xa. Một ứng dụng mạng có thể sử dụng cùng một IP endpoint cục bộ nhưng mỗi thiết bị ở xa phải sử dụng một địa chỉ IP hay port riêng.

IANA định nghĩa một danh sách các port TCP tiêu chuẩn được gán cho các ứng dụng đặc biệt:

Port	**Mô tả**
7	Echo
13	Daytime
17	Quote of the day
20	FTP (data channel)
21	FTP (control channel)
22	SSH
23	Telnet
25	SMTP
37	Time
80	HTTP
110	POP3
119	NNTP
123	Network Time Protocol (NTP)
137	NETBIOS name service
138	NETBIOS datagram service

143	Internet Message Access Protocol (IMAP)
389	Lightweight Directory Access Protocol (LDAP)
443	Secure HTTP (HTTPS)
993	Secure IMAP
995	Secure POP3

Các port từ 0->1023 được gán cho các ứng dụng thông dụng do đó với các ứng dụng mà các lập trình viên tạo ra thì các port được gán phải từ 1024->65535.

I.3.2. Cơ chế đảm bảo độ tin cậy truyền tải các gói tin

Trường tiếp theo trong TCP Header sau port là số sequence và acknowledgement. Những giá trị này cho phép TCP theo dõi các gói tin và đảm bảo nó được nhận theo đúng thứ tự. Nếu bất kỳ gói tin nào bị lỗi, TCP sẽ yêu cầu truyền tải lại các gói tin bị lỗi và ráp chúng lại trước khi gởi gói tin cho ứng dụng.

Mỗi gói tin có một số duy nhất sequence cho một phiên làm việc TCP. Một số ngẫu nhiên được chọn cho gói tin đầu tiên được gởi đi trong phiên làm việc. Mỗi gói tin tiếp theo được gởi sẽ tăng số sequence bằng số byte dữ liệu TCP trong gói tin trước đó. Điều này đảm bảo mỗi gói tin được xác định duy nhất trong luồng dữ liệu TCP.

Thiết bị nhận sử dụng trường acknowledgement để hồi báo số sequence cuối cùng được nhận từ thiết bị gởi. Thiết bị nhận có thể nhận nhiều gói tin trước khi gởi lại một hồi báo. Số acknowledgement được trả về là số sequence cao nhất liền sau của dữ liệu được nhận. Kỹ thuật này được gọi là cửa sổ trượt. Các gói tin được nhận ngoài thứ tự có thể được giữ trong bộ đệm và được đặt vào đúng thứ tự khi các gói tin khác đã được nhận thành công. Nếu một gói tin bị mất, thiết bị nhận sẽ thấy được số sequence bị lỗi và gởi một số acknowledgement thấp hơn để yêu cầu các gói tin bị lỗi. Nếu không có cửa sổ trượt mỗi gói tin sẽ phải hồi báo lại, làm tăng băng thông và độ trễ mạng.

I.3.3. Quá trình thành lập một phiên làm việc TCP

Quá trình thành lập một phiên làm việc TCP thực hiện nhờ vào việc sử dụng các cờ (Flag):

Flag	**Mô tả**
6 bit dành riêng	Dành riêng để sử dụng trong tương lai, giá trị luôn luôn là zero
1-bit URG flag	Đánh dấu gói tin là dữ liệu khẩn cấp
1-bit ACK flag	Hồi báo nhận một gói tin
1-bit PUSH flag	Cho biết dữ liệu được đẩy vào ứng dụng ngay lập tức
1-bit RESET flag	Thiết lập lại tình trạng khởi đầu kết nối TCP
1-bit SYN flag	Bắt đầu một phiên làm việc
1-bit FIN flag	Kết thúc một phiên làm việc

TCP sử dụng các tình trạng kết nối để quyết định tình trạng kết nối giữa các thiết bị. Một giao thức bắt tay đặc biệt được dùng để thành lập những kết nối này và theo dõi tình trạng kết nối trong suốt phiên làm việc. Một phiên làm việc TCP gồm ba pha sau:

• Mở bắt tay

• Duy trì phiên làm việc

• Đóng bắt tay

Mỗi pha yêu cầu các bit cờ được thiết lập trong một thứ tự nào đó. Quá trình mở bắt tay thường được gọi là ba cái bắt tay và nó yêu cầu ba bước để thành lập kết nối:

• Thiết bị gởi gởi cờ SYN cho biết bắt đầu phiên làm việc

• Thiết bị nhận gởi cả cờ SYN và cờ ACK trong cùng một gói tin cho biết nó chấp nhận bắt đầu phiên làm việc

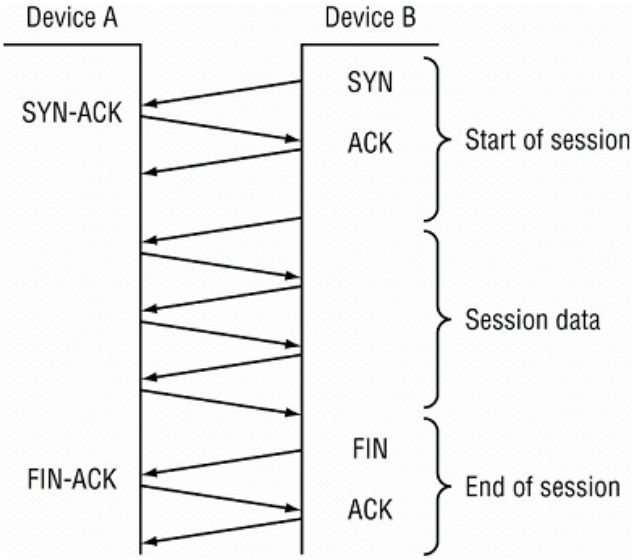
• Thiết bị gởi gởi cờ ACK cho biết phiên làm việc đã mở và đã sẵng sàng cho việc gởi và nhận các gói tin

Sau khi phiên làm việc được thành lập, cờ ACK sẽ được thiết lập trong các gói tin. Để đóng phiên làm việc, một quá trình bắt tay khác được thực hiện dùng cờ FIN:

• Thiết bị khởi đầu đóng kết nối gởi cờ FIN

• Thiết bị bên kia gởi cờ FIN và ACK trong cùng một gói tin cho biết nó chấp nhận đóng kết nối

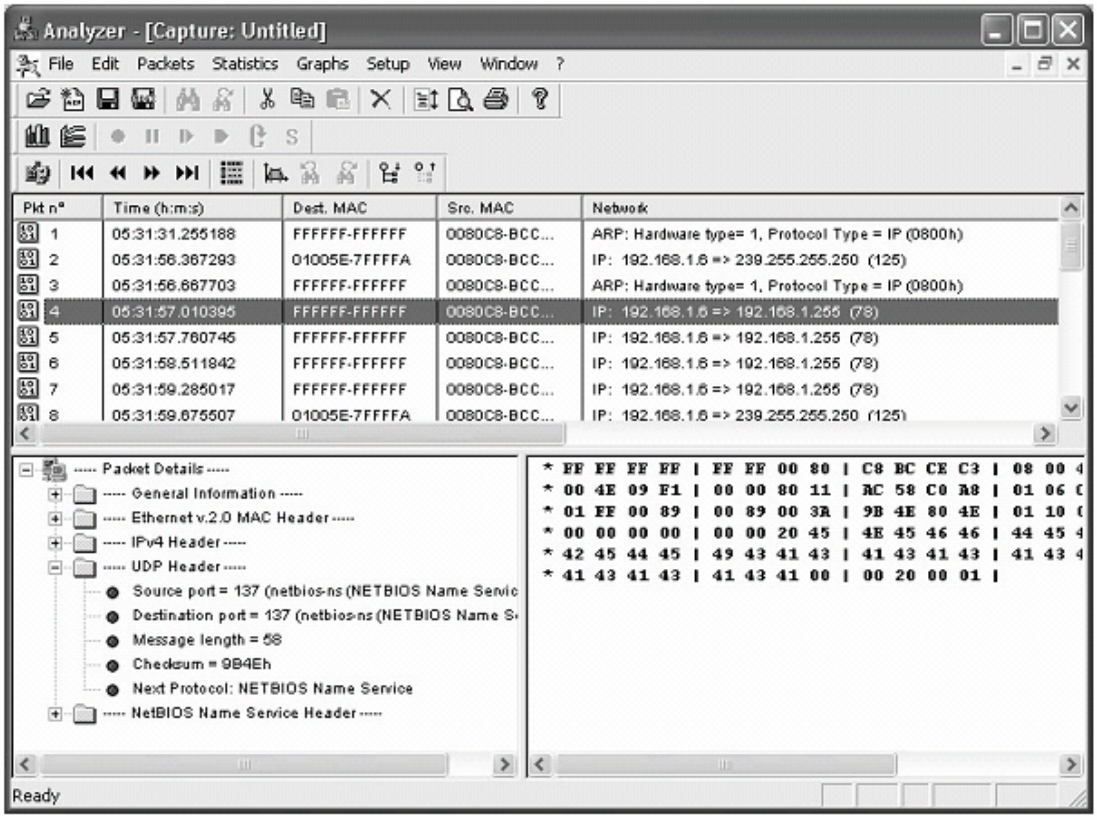
• Thiết bị khởi đầu đóng kết nối gởi cờ ACK để đóng kết nối



Hình I.6: Các bước bắt tay của giao thức TCP

I.4. Tầng UDP

User Datagram Protocol (UDP) là một giao thức phổ biến khác được dùng trong việc truyền tải dữ liệu của các gói IP. Không giống như TCP, UDP là giao thức phi nối kết. Mỗi phiên làm việc UDP không gì khác hơn là truyền tải một gói tin theo một hướng. Hình sau sẽ mô tả cấu trúc của một gói tin UDP



Hình I.7: UDP Header

UDP header gồm những trường sau:

- Source Port
- Destination Port
- Message Length
- Checksum
- Next Level Protocol

Cũng giống như TCP, UDP theo dõi các kết nối bằng cách sử dụng các port từ 1024->65536, các port UDP từ 0->1023 là các port dành riêng cho các ứng dụng phổ biến, một số dùng phổ biến như:

Port	**Mô tả**
53	Domain Name System
69	Trivial File Transfer Protocol
111	Remote Procedure Call
137	NetBIOS name service
138	NetBIOS datagram
161	Simple Network Management Protocol

CHƯƠNG 2. LẬP TRÌNH SOCKET HƯỚNG KẾT NỐI

II.1. Socket

Trong lập trình mạng dùng Socket, chúng ta không trực tiếp truy cập vào các thiết bị mạng để gởi và nhận dữ liệu. Thay vì vậy, một file mô tả trung gian được tạo ra để điều khiển việc lập trình. Các file mô tả dùng để tham chiếu đến các kết nối mạng được gọi là các Socket. Socket định nghĩa những đặc trưng sau:

• Một kết nối mạng hay một đường ống dẫn để truyền tải dữ liệu

• Một kiểu truyền thông như stream hay datagram

• Một giao thức như TCP hay UDP

Sau khi một Socket được tạo ra nó phải được gắn vào một địa chỉ mạng và một port trên hệ thống cục bộ hay ở xa. Một khi Socket đã được gắn vào các địa chỉ mạng và port, nó có thể được dùng để gởi và nhận dữ liệu trong mạng.

Trong .Net Framework lớp Socket hỗ trợ cho việc lập trình Socket. Phương thức tạo lập như sau: Socket (AddressFamily, SocketType, ProtocolType)

Phương thức tạo lập của lớp Socket cần các đối số truyền vào sau:

- **AddressFamily:** họ địa chỉ được dùng, tham số này có thể có các giá trị sau:

Kiểu	**Mô tả**
AppleTalk	Địa chỉ AppleTalk
Atm	Native ATM services address.
Banyan	Địa chỉ Banyan
Ccitt	Địa chỉ cho giao thức CCITT, như là X25
Chaos	Địa chỉ cho giao thức MIT CHAOS
Cluster	Địa chỉ cho các sản phẩm cluster của Microsoft
DataKit	Địa chỉ cho giao thức Datakit
DataLink	Địa chỉ của giao thức tầng data-link
DecNet	Địa chỉ DECnet
Ecma	Địa chỉ ECMA (European Computer Manufacturers Association)
FireFox	Địa chỉ FireFox
HyperChannel	Địa chỉ NSC Hyperchannel
Ieee12844	Địa chỉ workgroup IEEE 1284.4
ImpLink	Địa chỉ ARPANET IMP
InterNetwork	Địa chỉ IP version 4
InterNetworkV6	Địa chỉ IP version 6
Ipx	Địa chỉ IPX hoặc SPX
Irda	Địa chỉ IrDA

Iso	Địa chỉ cho giao thức ISO
Lat	Địa chỉ LAT
Max	Địa chỉ MAX
NetBios	Địa chỉ NetBios
NetworkDesigners	Địa chỉ Network Designers
NS	Địa chỉ Xerox NS
Osi	Địa chỉ cho giao thức ISO
Pup	Địa chỉ cho giao thức PUP
Sna	Địa chỉ IBM SNA
Unix	Địa chỉ Unix
Unknown	Chưa biết họ địa chỉ
Unspecified	Chưa chỉ ra họ địa chỉ
VoiceView	Địa chỉ VoiceView

- SocketType: kiểu Socket, tham số này có thể có các giao thức sau:

Kiểu	**Mô tả**
Dgram	Được sử dụng trong các giao thức phi kết nối, không tin tưởng. Thông điệp có thể bị mất, bị trùng lặp hoặc có thể đến sai thứ tự. Dgram sử dụng giao thức UDP và họ địa chỉ InterNetwork.
Raw	Được sử trong các giao thức cấp thấp như Internet Control Message Protocol (Icmp) và Internet Group Management Protocol (Igmp). Ứng dụng phải cung cấp IP header khi gởi. Khi nhận sẽ nhận được IP header và các tùy chọn tương ứng.
Rdm	Được sử dụng trong các giao thức phi kết nối, hướng thông điệp, truyền thông điệp tin cậy, và biên của thông điệp được bảo vệ. Rdm (Reliably Delivered Messages) thông điệp đến không bị trùng lặp và đúng thứ tự. Hơn nữa, thiết bị nhận được thiết bị nếu thông điệp bị mất. Nếu khởi tạo Socket dùng Rdm, ta không cần yêu cầu kết nối tới host ở xa trước khi gởi và nhận dữ liệu.
Seqpacket	Cung cấp hướng kết nối và truyền 2 chiều các dòng byte một cách tin cậy. Seqpacket không trùng lập dữ liệu và bảo vệ biên dữ liệu. Socket kiểu Seqpacket truyền thông với 1 máy đơn và yêu cầu kết nối trước khi truyền dữ liệu.
Stream	Được sử dụng trong các giao thức hướng kết nối, không bị trùng lặp dữ liệu, không bảo vệ biên dữ liệu. Socket kiểu Stream chỉ truyền thông với một máy đơn và yêu cầu kết nối trước khi truyền dữ liệu. Stream dùng giao thức Transmission Control Protocol (Tcp) và họ địa chỉ InterNetwork.

Unknown	Chưa biết kiểu Socket

- ProtocolType: kiểu giao thức, tham số này có thể có các giá trị sau:

ProtocolType	**Mô tả**
Ggp	Gateway To Gateway Protocol.
Icmp	Internet Control Message Protocol.
IcmpV6	Internet Control Message Protocol IPv6.
Idp	Internet Datagram Protocol.
Igmp	Internet Group Management Protocol.
IP	Internet Protocol.
IPSecAuthenticationHeader	IPv6 Authentication.
IPSecEncapsulatingSecurityPayload	IPv6 Encapsulating Security Payload header.
IPv4	Internet Protocol version 4.
IPv6	Internet Protocol version 6 (IPv6).
Ipx	Internet Packet Exchange Protocol.
ND	Net Disk Protocol (unofficial).
Pup	PARC Universal Packet Protocol.
Raw	Raw IP packet protocol.
Spx	Sequenced Packet Exchange protocol.
SpxII	Sequenced Packet Exchange version 2 protocol.
Tcp	Transmission Control Protocol.
Udp	User Datagram Protocol.
Unknown	Giao thức chưa biết
Unspecified	Giao thức chưa được chỉ ra

Ví dụ phương thức tạo lập của lớp Socket:

```
Socket sk = Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

II.2. IPAddress

IPAddress là một đối tượng dùng để mô tả một địa chỉ IP, đối tượng này có thể được sử dụng trong nhiều phương thức của Socket. Một số phương thức của lớp IPAddress

Phương thức	**Mô tả**
Equals	So sánh 2 địa chỉ IP
GetHashCode	Lấy giá trị hash cho 1 đối tượng IPAddress
GetType	Trả về kiểu của một thể hiện địa chỉ IP
HostToNetworkOrder	Chuyển 1 địa chỉ IP từ host byte order thành network byte order IsLoopBack. Cho biết địa chỉ IP có phải là địa chỉ LoopBack hay không

NetworkToHostOrder	Chuyển 1 địa chỉ IP từ network byte order thành host byte order
Parse	Chuyển 1 chuỗi thành 1 thể hiện IPAddress
ToString	Chuyển 1 đối tượng IPAddress thành một chuỗi

Phương thức Parse() thường được dùng để tạo ra 1 thể hiện của IPAddress:

```
IPAddress localIpAddress = IPAddress.Parse("127.0.0.1");
```

Lớp IPAddress cũng cung cấp 4 thuộc tính để mô tả các địa chỉ IP đặc biệt:

- Any: dùng để mô tả một địa chỉ IP bất kỳ của hệ thống.
- Broadcast: dùng để mô tả địa chỉ IP Broadcast cho mạng cục bộ
- Loopback: dùng để mô tả địa chỉ loopback của hệ thống
- None: không dùng địa chỉ IP

II.3. IPEndPoint

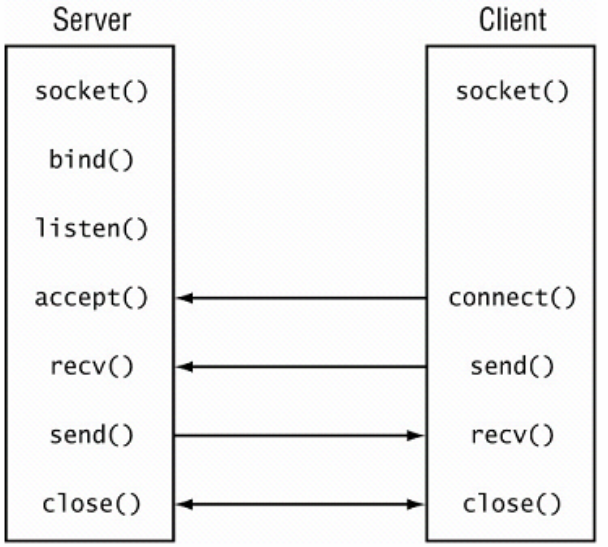
IPEndPoint là một đối tượng mô tả sự kết hợp của một địa chỉ IP và port. Đối tượng IPEndPoint được dùng để gắn kết các Socket với các địa chỉ cục bộ hoặc các địa chỉ ở xa.

Hai thuộc tính của IPEndPoint có thể được dùng để lấy được vùng các port trên hệ thống là MinPort và MaxPort.

II.4. Lập trình Socket hướng kết nối

Trong lập trình Socket hướng kết nối, giao thức TCP được dùng để thành lập phiên làm việc giữa hai endpoint. Khi sử dụng giao thức TCP để thành lập kết nối ta phải đàm phán kết nối trước nhưng khi kết nối đã được thành lập dữ liệu có thể truyền đi giữa các thiết bị một cách tin tưởng.

Để lập trình Socket hướng kết nối ta phải thực hiện một loạt các thao tác giữa client và Server như trong mô hình bên dưới



Hình II.1: Mô hình lập trình Socket hướng kết nối

II.4.1. Lập trình phía Server

Đầu tiên Server sẽ tạo một Socket, Socket này sẽ được gắn vào một địa chỉ ip và một port cục bộ, hàm để thực hiện việc này là hàm Bind(). Hàm này cần một danh đối số là một IPEndPoint cục bộ:

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
server.Bind(ipep);
```

Bởi vì Server thường chấp nhận kết nối trên chính địa chỉ IP và port riêng của nó nên ta dùng IPAddress.Any để chấp nhận kết nối trên bất kỳ card mạng nào.

Địa chỉ IP ta dùng ở đây là địa chỉ IP version 4 và kiểu giao thức là TCP nên AddressFamily là InterNetwork và SocketType là Stream.

Sau khi Socket đã được gắn kết vào một địa chỉ và một port, Server phải sẵn sàng chấp nhận kết nối từ Client. Việc này được thực hiện nhờ vào hàm Listen(). Hàm Listen() có một đối số, đó chính là số Client tối đa mà nó lắng nghe.

```
server.Listen(10);
```

Tiếp theo Server dùng hàm Accept() để chấp nhận kết nối từ Client:

```
Socket client = server.Accept();
```

Hàm Accept() này sẽ dừng Server lại và chờ cho đến khi nào có Client kết nối đến nó sẽ trả về một Socket khác, Socket này được dùng để trao đổi dữ liệu với Client. Khi đã chấp nhận kết nối với Client thì Server có thể gởi và nhận dữ liệu với Client thông qua phương thức Send() và Receive().

```
string welcome = "Hello Client";
buff = Encoding.ASCII.GetBytes(welcome);
client.Send(buff, buff.Length, SocketFlags.None);
```

Phương thức Send() của Socket dùng để gởi dữ liệu, phương thức này có một số đối số quan trọng sau:

- Buff : mảng các byte cần gởi
- Offset: vị trí đầu tiên trong mảng cần gởi
- Size: số byte cần gởi
- SocketFlags: chỉ ra cách gởi dữ liệu trên Socket

Việc gởi và nhận dữ liệu được thực hiện liên tục thông qua một vòng lặp vô hạn:

```
while (true)
{
        buff = new byte[1024];
        recv = client.Receive(buff);
        if (recv == 0)
```

```
            break;
        Console.WriteLine(Encoding.ASCII.GetString(buff, 0, recv));
        client.Send(buff, recv, SocketFlags.None);
    }
```

Phương thức Receive() đặt dữ liệu vào buffer, kích thước buffer được thiết lập lại, do đó nếu buffer không được thiết lập lại, lần gọi phương thức Receive() kế tiếp sẽ chỉ có thể nhận được dữ liệu tối đa bằng lần nhận dữ liệu trước. Phương thức này có một số đối số quan trọng sau:

- Buff : mảng các byte cần gởi
- Offset: vị trí đầu tiên trong mảng cần nhận
- Size: số byte cần gởi
- SocketFlags: chỉ ra cách nhận dữ liệu trên Socket

Phương thức Receive() trả về số byte dữ liệu nhận được từ Client. Nếu không có dữ liệu được nhận, phương thức Receive() sẽ bị dừng lại và chờ cho tới khi có dữ liệu. Khi Client gởi tín hiệu kết thúc phiên làm việc (bằng cách gởi cờ FIN trong gói TCP), phương thức Receive() sẽ trả về giá trị 0. Khi phương thức Receive() trả về giá trị 0, ta đóng Socket của Client lại bằng phương thức Close(). Socket chính (Server Socket) vẫn còn hoạt động để chấp nhận các kết nối khác. Nếu không muốn Client nào kết nối đến nữa thì ta đóng Server lại luôn:

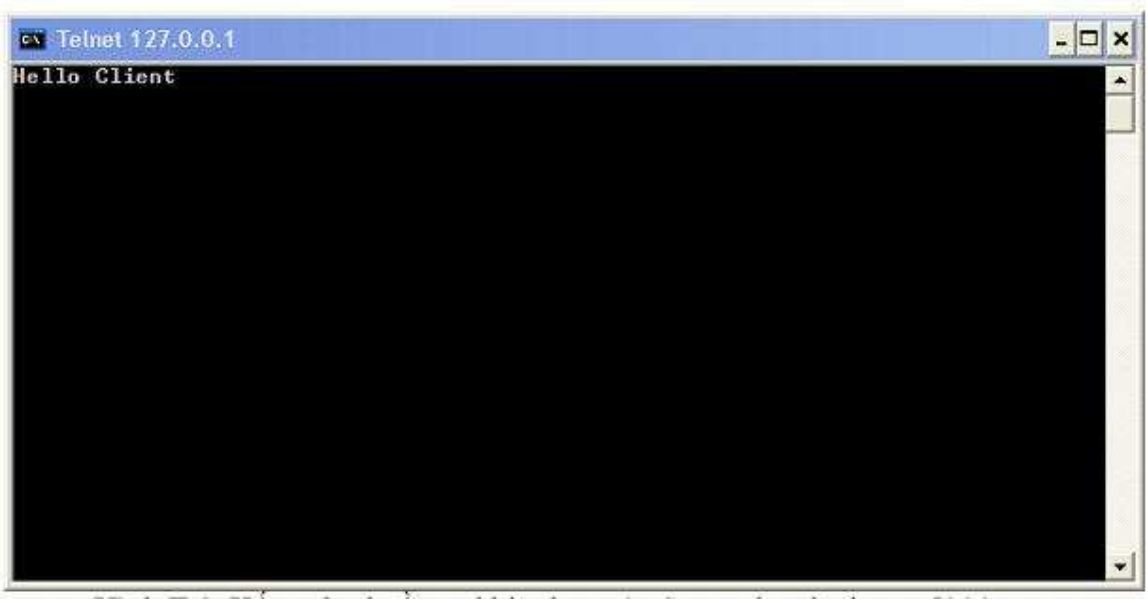
```
client.Close();
server.Close();
```

Chương trình TCP Server đơn giản:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpServerDonGian
{
    public static void Main()
    {
        //Số byte thực sự nhận được dùng hàm Receive()
        int byteReceive;
        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //EndPoint cục bộ
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
```

```
            //Kết nối server với 1 EndPoint
            server.Bind(ipep);
            //Server lắng nghe tối đa 10 kết nối
            server.Listen(10);
            Console.WriteLine("Dang cho Client ket noi den...");
            //Hàm Accept() sẽ block server lại cho đến khi có Client kết nối đến
            Socket client = server.Accept();
            //Client EndPoint
            IPEndPoint clientep = (IPEndPoint)client.RemoteEndPoint;
            Console.WriteLine("Da ket noi voi Client {0} tai port {1}", clientep.Address,
clientep.Port);
            string welcome = "Hello Client";
            //Chuyển chuỗi thành mảng các byte
            buff = Encoding.ASCII.GetBytes(welcome);
            //Gởi câu chào cho Client
            client.Send(buff, buff.Length, SocketFlags.None);
            while (true)
            {
                //Reset lại buffer
                buff = new byte[1024];
                //Lấy số byte thực sự nhận được
                byteReceive = client.Receive(buff);
                //Nếu Client ngắt kết nối thì thoát khỏi vòng lặp
                if (byteReceive == 0) break;
                Console.WriteLine(Encoding.ASCII.GetString(buff, 0, byteReceive));
                //Sau khi nhận dữ liệu xong, gởi lại cho Client
                client.Send(buff, byteReceive, SocketFlags.None);
            }
            Console.WriteLine("Da dong ket noi voi Client: {0}", clientep.Address);
            //Đóng kết nối
            client.Close();
            server.Close();
        }
}
```

Để kiểm tra thử chương trình ta có thể dùng chương trình Telnet của Windows để kiểm tra. Dùng lệnh telnet 127.0.0.1 5000



Hình II.2: Kết quả trả về sau khi telnet vào Server local tại port 5000

Sau khi dùng lệnh telnet, kết quả trả về như trên hình là đã kết nối thành công.

II.4.2. Lập trình phía Client

Lập trình Socket hướng kết nối phía Client đơn giản hơn phía Server. Client cũng phải gắn kết một địa chỉ của một Socket đã được tạo ra nhưng sử dụng phương thức Connect() chứ không sử dụng phương thức Bind() giống như phía Server. Phương thức Connect() yêu cầu một IPEndPoint của Server mà Client cần kết nối đến.

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
try
{
        server.Connect(ipep);
}
catch (SocketException e)
{
        Console.WriteLine("Không thể kết nối đến Server");
        Console.WriteLine(e.ToString());
        return;
}
```

Phương thức Connect() sẽ dừng lại cho đến khi Client kết nối được với Server. Nếu kết nối không thể được thực hiện thì nó sẽ phát sinh ra một biệt lệ, do đó hàm Connect() tra phải để trong khối try, catch để không bị lỗi chương trình.

Khi kết nối được thành lập, Client có thể dùng phương thức Send() và Receive() của lớp Socket để gởi và nhận dữ liệu tương tự như Server đã làm. Khi quá trình trao đổi dữ liệu đã hoàn tất, đối tượng Socket phải được đóng lại. Client Socket dùng phương thức Shutdown() để dừng Socket và dùng phương thức Close() để thực sự đóng phiên làm việc. Phương thức Shutdown() của Socket dùng một tham số để quyết định cách Socket sẽ dừng lại. Các phương thức đó là:

Giá trị	**Mô tả**
SocketShutdown.Both	Ngăn cản gởi và nhận dữ liệu trên Socket.
SocketShutdown.Receive	Ngăn cản nhận dữ liệu trên Socket. Cờ RST sẽ được gởi nếu có thêm dữ liệu được nhận.
SocketShutdown.Send	Ngăn cản gởi dữ liệu trên Socket. Cờ FIN sẽ được gởi sau khi tất cả dữ liệu còn lại trong buffer đã được gởi đi.

Chương trình TCP Client đơn giản:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleTcpClient
{
    public static void Main()
    {
        //Buffer để gởi và nhận dữ liệu
        byte[] buff = new byte[1024];
        //Chuỗi nhập vào và chuỗi nhận được
        string input, stringData;
        //IPEndPoint ở server
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        //Hàm Connect() sẽ block lại và chờ đến khi kết nối với server thì hết block
        try
        {
            server.Connect(ipep);
        }
```

```
            //Quá trình kết nối có thể xảy ra lỗi nên phải dùng try, catch
            catch (SocketException e)
            {
                Console.WriteLine("Không thể kết nối đến Server");
                Console.WriteLine(e.ToString());
                return;
            }
            //Số byte thực sự nhận được
            int byteReceive = server.Receive(buff);
            //Chuỗi nhận được
            stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
            Console.WriteLine(stringData);
            while (true)
            {
                //Nhập dữ liệu từ bàn phím
                input = Console.ReadLine();
                //Nếu nhập exit thì thoát và đóng Socket
                if (input == "exit") break;
                //Gởi dữ liệu cho server
                server.Send(Encoding.ASCII.GetBytes(input));
                //Reset lại buffer
                buff = new byte[1024];
                //Số byte thực sự nhận được
                byteReceive = server.Receive(buff);
                //Chuỗi nhận được
                stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
                Console.WriteLine(stringData);
            }
            Console.WriteLine("Dong ket noi voi server...");
            //Dừng kết nối, không cho phép nhận và gởi dữ liệu
            server.Shutdown(SocketShutdown.Both);
            //Đóng Socket
            server.Close();
        }
    }
```

II.4.3. Vấn đề với bộ đệm dữ liệu

Trong ví dụ Client, Server đơn giản trên thì một mảng các byte được dùng như bộ đệm để gởi và nhận dữ liệu trên Socket. Bởi vì chương trình được chạy trong môi trường được điều khiển, tất cả các thông điệp đều thuộc dạng text và kích thước nhỏ nên loại buffer này không phải là một vấn đề.

Trong thực tế, chúng ta không biết kích thước và kiểu dữ liệu đến trong khi truyền thông giữa Client và Server. Vấn đề xảy ra khi dữ liệu đến lớn hơn kích thước bộ đệm dữ liệu.

Khi nhận dữ liệu thông qua TCP, dữ liệu được lưu trữ trong bộ đệm hệ thống. Mỗi khi gọi phương thức Receive(), nó sẽ đọc dữ liệu từ bộ đệm TCP và lấy dữ liệu ra khỏi bộ đệm. Số lượng dữ liệu được đọc bởi phương thức Receive() được điều khiển bởi hai yếu tố sau:

• Kích thước bộ đệm dữ liệu được chỉ ra trong phương thức Receive()

• Kích thước bộ đệm được chỉ ra trong tham số của phương thức Receive()

Trong ví dụ đơn giản trên, buffer được định nghĩa là một mảng byte kích thước 1024. Bởi vì kích thước dữ liệu không được chỉ ra trong phương thức Receive() nên kích thước bộ đệm tự động lấy kích thước mặc định của bộ đệm dữ liệu là 1024 byte. Phương thức Receive() sẽ đọc 1024 byte dữ liệu một lần và đặt dữ liệu đọc được vào biến buff như sau:

```
byteReceive = client.Receive(buff);
```

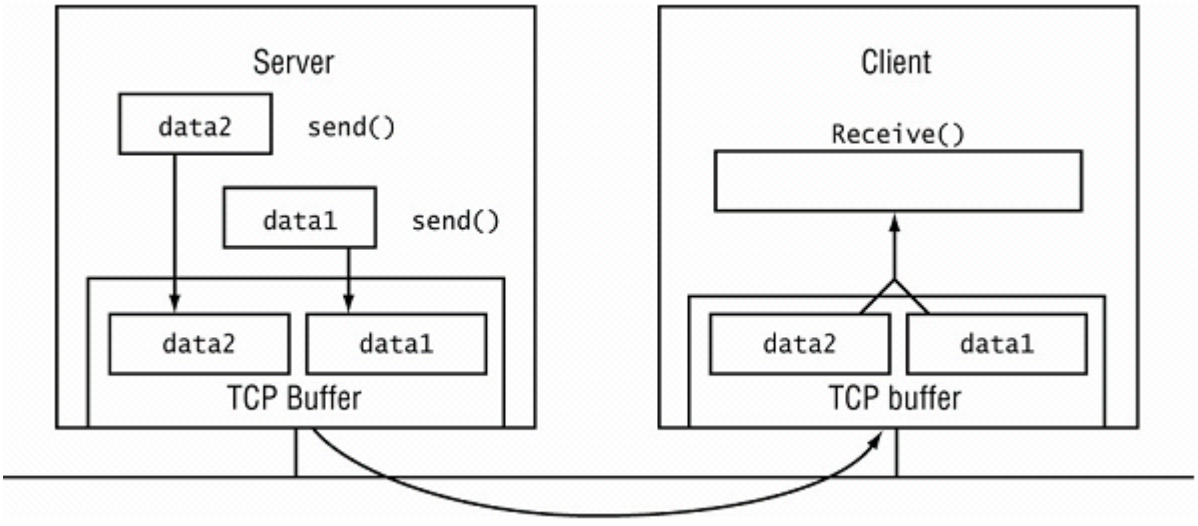
Vào lúc phương thức Receive() được gọi, nếu bộ đệm TCP chứa ít hơn 1024 byte, phương thức này sẽ trả về số lượng dữ liệu mà nó thực sự đọc được trong biến byte Receive. Để chuyển dữ liệu thành chuỗi, ta dùng phương thức GetString() như sau:

```
stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
```

Trong đối số của hàm GetString, ta phải truyền vào số byte thực sự đã đọc được nếu không ta sẽ nhận được một chuỗi với các byte thừa ở đằng sau.

II.4.4. Xử lý với các bộ đệm có kích thước nhỏ

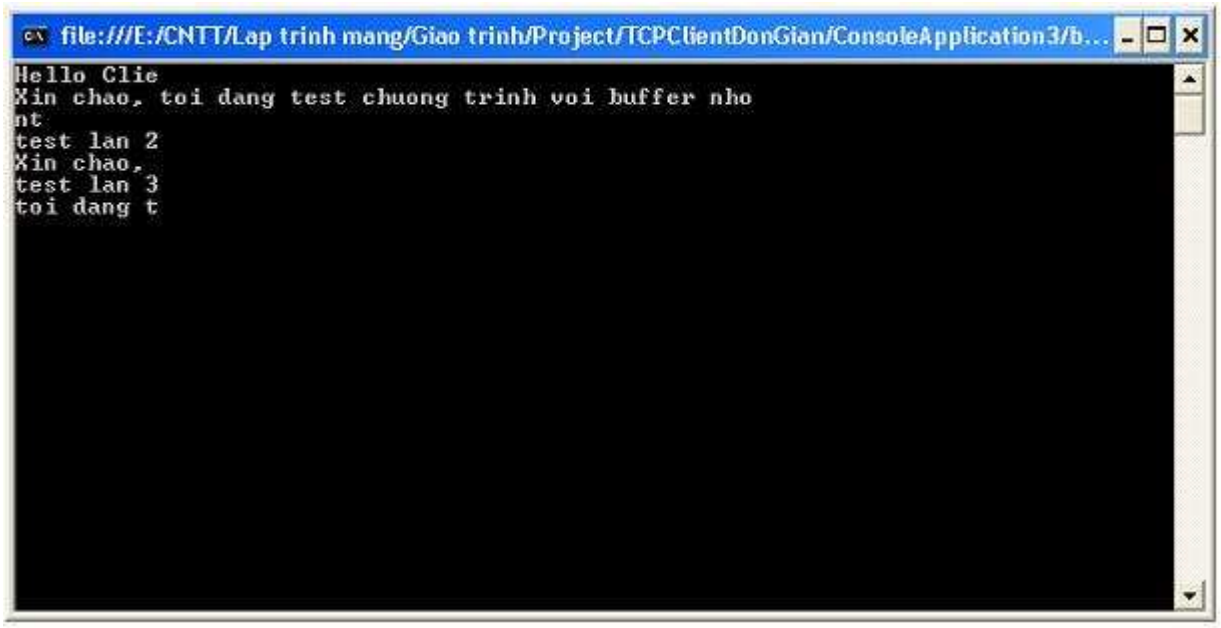
Hệ điều hành Window dùng bộ đệm TCP để gởi và nhận dữ liệu. Điều này là cần thiết để TCP có thể gởi lại dữ liệu bất cứ lúc nào cần thiết. Một khi dữ liệu đã được hồi báo nhận thành công thì nó mới được xóa khỏi bộ đệm.



Hình II.3: TCP Buffer

Dữ liệu đến cũng được hoạt động theo cách tương tự. Nó sẽ ở lại trong bộ đệm cho đến khi phương thức Receive() được dùng để đọc nó. Nếu phương thức Receive() không đọc toàn bộ dữ liệu ở trong bộ đệm, phần còn lại vẫn được nằm ở đó và chờ phương thức Receive() tiếp theo được đọc. Dữ liệu sẽ không bị mất nhưng chúng ta sẽ không lấy được các đoạn dữ liệu mình mong muốn.

Để thấy được vấn đề, ta tiến hành thay đổi kích thước bộ đệm từ 1024 byte xuống còn 10 byte. Và chạy lại chương trình Client, Server đơn giản trên



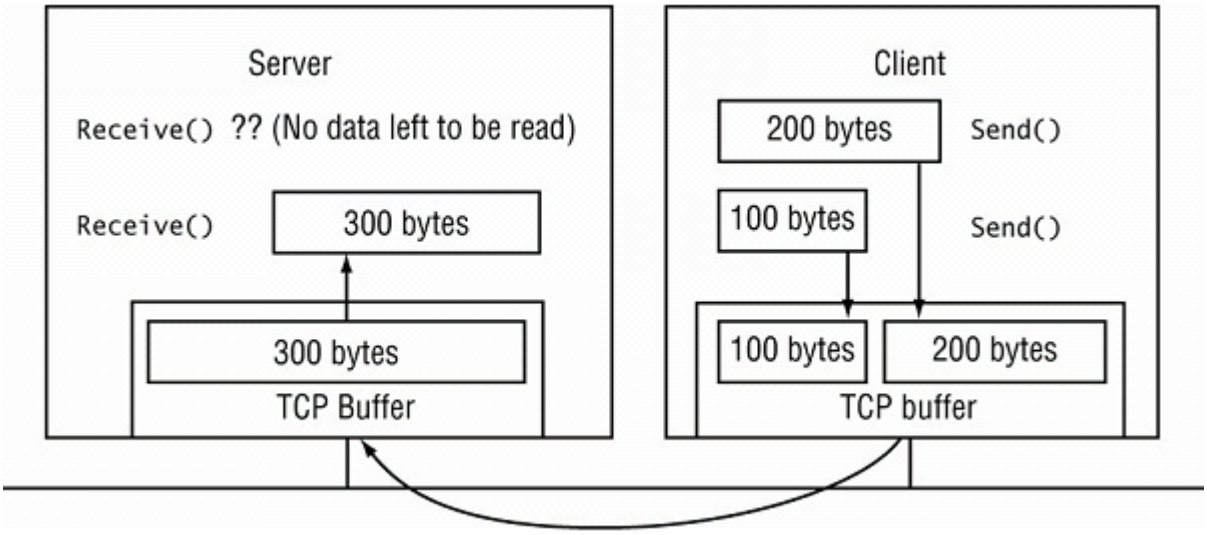
Hình II.4: Kết quả trả về khi chạy chương trình với buffer nhỏ

Bởi vì bộ đệm dữ liệu không đủ lớn để lấy hết dữ liệu ở bộ đệm TCP nên phương thức Receive() chỉ có thể lấy được một lượng dữ liệu có độ lớn đúng bằng độ lớn của bộ đệm dữ liệu, phần còn lại vẫn nằm ở bộ đệm TCP và nó được lấy khi gọi lại phương thức Receive(). Do đó câu chào Client của Server phải dùng tới hai lần gọi phương thức Receive() mới lấy được hết. Trong lần gởi và nhận dữ liệu kế tiếp, đoạn dữ liệu tiếp theo được đọc từ bộ đệm TCP do đó nếu ta gởi dữ liệu với kích thước lớn hơn 10 byte thì khi nhận ta chỉ nhận được 10 byte đầu tiên.

Vì vậy, trong khi lập trình mạng chúng ta phải quan tâm đến việc đọc dữ liệu từ bộ đệm TCP một cách chính xác. Bộ đệm quá nhỏ có thể dẫn đến tình trạng thông điệp nhận sẽ không khớp với thông điệp gởi, ngược lại bộ đệm quá lớn sẽ làm cho các thông điệp bị trộn lại, khó xử lý. Việc khó nhất là làm sao phân biệt được các thông điệp được đọc từ Socket.

II.4.5. Vấn đề với các thông điệp TCP

Một trong những khó khăn của những nhà lập trình mạng khi sử dụng giao thức TCP để chuyển dữ liệu là giao thức này không quan tâm đến biên dữ liệu.



Hình II.5: Client Send hai lần rồi Server mới Receive

Như trên hình vấn đề xảy ra khi truyền dữ liệu là không đảm bảo được mỗi phương thức Send() sẽ không được đọc bởi một phương thức Receive(). Tất cả dữ liệu được đọc từ phương thức Receive() không thực sự được đọc trực tiếp từ mạng mà nó được đọc từ bộ đệm TCP. Khi các gói tin TCP được nhận từ mạng sẽ được đặt theo thứ tự trong bộ đệm TCP. Mỗi khi phương thức Receive() được gọi, nó sẽ đọc dữ liệu trong bộ đệm TCP, không quan tâm đến biên dữ liệu. Chúng ta hãy xem xét ví dụ sau:

Chương Trình BadTCPServer:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BadTcpServer
{
    public static void Main()
    {
        //Số byte thực sự nhận được dùng hàm Receive()
        int byteReceive;
        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //EndPoint cục bộ
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        //Kết nối server với 1 EndPoint
```

```
                server.Bind(ipep);
                //Server lắng nghe tối đa 10 kết nối
                server.Listen(10);
                Console.WriteLine("Dang cho Client ket noi den...");
                //Hàm Accept() sẽ block server lại cho đến khi có Client kết nối đến
                Socket client = server.Accept();
                //Client EndPoint
                IPEndPoint clientep = (IPEndPoint)client.RemoteEndPoint;
                Console.WriteLine("Da ket noi voi Client {0} tai port {1}", clientep.Address,
                clientep.Port);
                string welcome = "Hello Client";
                //Chuyển chuỗi thành mảng các byte
                buff = Encoding.ASCII.GetBytes(welcome);
                //Gởi câu chào cho Client
                client.Send(buff, buff.Length, SocketFlags.None);
                for (int i = 0; i < 5; i++)
                {
                        byteReceive = client.Receive(buff);
                        Console.WriteLine(Encoding.ASCII.GetString(buff, 0, byteReceive));
                }
                Console.WriteLine("Da dong ket noi voi Client: {0}", clientep.Address);
                //Đóng kết nối
                client.Close();
                server.Close();
                Console.Read();
        }
}
```

Chương trình Server thành lập Socket TCP bình thường để lắng nghe kết nối, khi kết nối được thành lập, Server gởi câu chào cho Client và cố gắng nhận năm thông điệp riêng biệt từ Client:

```
for (int i = 0; i < 5; i++)
{
        byteReceive = client.Receive(data);
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, byteReceive));
}
```

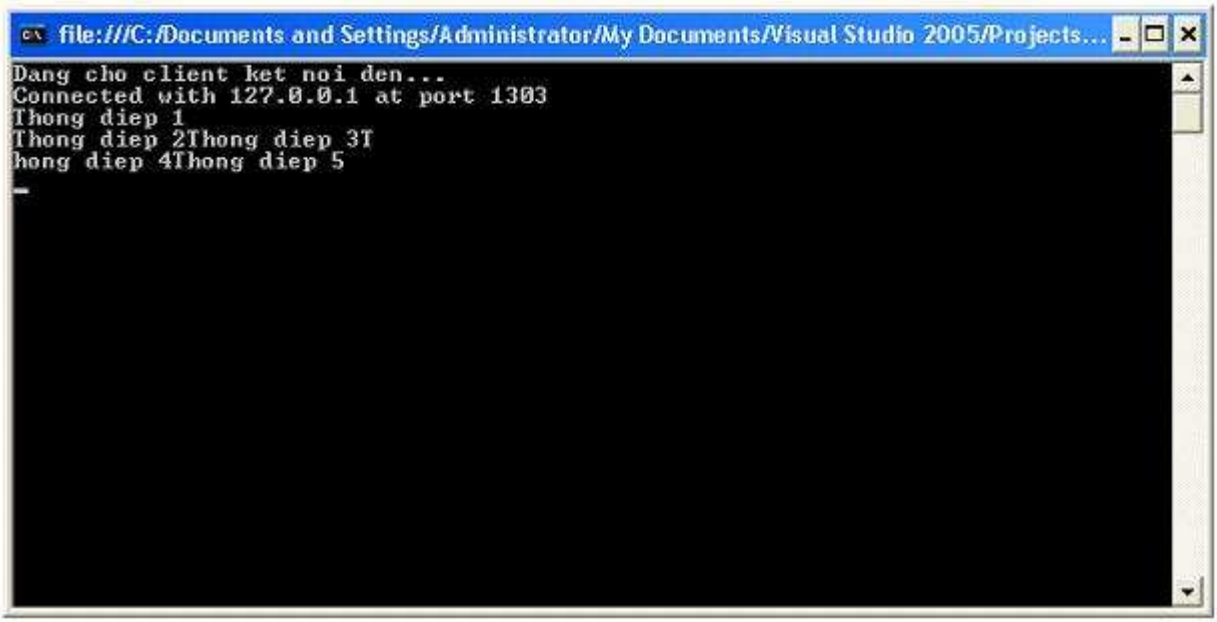
Mỗi khi được gọi, phương thức Receive() đọc toàn bộ dữ liệu trong bộ đệm TCP, sau khi nhận năm thông điệp, kết nối được đóng lại.

Chương trình BadTCPClient:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BadTcpClient
{
    public static void Main()
    {
        //Buffer để gởi và nhận dữ liệu
        byte[] buff = new byte[10];
        //Chuỗi nhập vào và chuỗi nhận được
        string input, stringData;
        //IPEndPoint ở server
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        //Hàm Connect() bị block lại và chờ đến khi kết nối với server thì hết block
        try
        {
            server.Connect(ipep);
        }
        //Quá trình kết nối có thể xảy ra lỗi nên phải dùng try, catch
        catch (SocketException e)
        {
            Console.WriteLine("Khong the ket noi den Server");
            Console.WriteLine(e.ToString());
            return;
        }
        //Số byte thực sự nhận được
        int byteReceive = server.Receive(buff);
        //Chuỗi nhận được
        stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
        Console.WriteLine(stringData);
```

```
                server.Send(Encoding.ASCII.GetBytes("Thong diep 1"));
                server.Send(Encoding.ASCII.GetBytes("Thong diep 2"));
                server.Send(Encoding.ASCII.GetBytes("Thong diep 3"));
                server.Send(Encoding.ASCII.GetBytes("Thong diep 4"));
                server.Send(Encoding.ASCII.GetBytes("Thong diep 5"));
                Console.WriteLine("Dong ket noi voi server...");
                //Dừng kết nối, không cho phép nhận và gởi dữ liệu
                server.Shutdown(SocketShutdown.Both);
                //Đóng Socket
                server.Close();
                Console.Read();
            }
        }
```

Kết quả chương trình như hình bên dưới



Hình II.6: Kết quả trên Server

Trong lần gọi phương thức Receive() lần đầu tiên, phương thức này nhận toàn bộ dữ liệu từ phương thức Send() của Client gởi lên, trong lần gọi phương thức Receive() lần thứ hai, phương thức Receive() đọc dữ liệu từ hai phương thức Send() và một phương thức Send() khác gởi dữ liệu chưa xong. Trong lần gọi thứ ba thì phương thức Receive() sẽ đọc hết dữ liệu đang được gởi dở từ phương thức Send() và đọc dữ liệu được gởi từ phương thức Send() cuối cùng và sau khi Client thực hiện xong năm phương thức Send() nó sẽ đóng kết nối với Server và Server cũng sẽ thoát ra.

II.4.6. Giải quyết các vấn đề với thông điệp TCP

Để giải quyết vấn đề với biên dữ liệu không được bảo vệ, chúng ta phải tìm hiểu một số kỹ thuật để phân biệt các thông điệp. Ba kỹ thuật thông thường dùng để phân biệt các thông điệp được gởi thông qua TCP:

- Luôn luôn sử dụng các thông điệp với kích thước cố định
- Gởi kèm kích thước thông điệp cùng với mỗi thông điệp
- Sử dụng các hệ thống đánh dấu để phân biệt các thông điệp

II.4.6.1. Sử dụng các thông điệp với kích thước cố định

Cách dễ nhất nhưng cũng là cách tốn chi phí nhất để giải quyết vấn đề với các thông điệp TCP là tạo ra các giao thức luôn luôn truyền các thông điệp với kích thước cố định. Bằng cách thiết lập tất cả các thông điệp có cùng kích thước, chương trình TCP nhận có thể biết toàn bộ thông điệp được gởi từ Client.

Khi gởi dữ liệu với kích thước cố định, chúng ta phải đảm bảo toàn bộ thông điệp được gởi từ phương thức Send(). Phụ thuộc vào kích thước của bộ đệm TCP và bao nhiêu dữ liệu được truyền, phương thức Send() sẽ trả về số byte mà nó thực sự đã gởi đến bộ đệm TCP. Nếu phương thức Send() chưa gởi hết dữ liệu thì chúng ta phải gởi lại phần dữ liệu còn lại. Việc này thường được thực hiện bằng cách sử dụng vòng lặp while() và trong vòng lặp ta kiểm tra số byte thực sự đã gởi với kích thước cố định.

```
private static int SendData(Socket s, byte[] data)
{
        int total = 0;
        int size = data.Length;
        int dataleft = size;
        int sent;
        while (total < size)
        {
                sent = s.Send(data, total, dataleft, SocketFlags.None);
                total += sent;
                dataleft -= sent;
        }
        return total;
}
```

Cũng giống như việc gởi dữ liệu, chúng ta phải luôn luôn đảm bảo nhận tất cả dữ liệu trong phương thức Receive(). Bằng cách dùng vòng lặp gọi phương thức Receive() chúng ta có thể nhận được toàn bộ dữ liệu mong muốn.

```
private static byte[] ReceiveData(Socket s, int size)
{
        int total = 0;
```

```
        int dataleft = size;
        byte[] data = new byte[size];
        int recv;
        while (total < size)
        {
            recv = s.Receive(data, total, dataleft, 0);
            if (recv == 0)
            {
                data = Encoding.ASCII.GetBytes("exit");
                break;
            }
            total += recv;
            dataleft -= recv;
        }
        return data;
    }
```

Phương thức ReceiveData() sẽ đọc dữ liệu với kích thước được đọc là đối số được truyền vào, nếu phương thức Receive() sẽ trả về số byte thực sụ đọc được, nếu số byte thực sự đọc được mà còn nhỏ hơn số byte truyền vào phương thức ReceiveData() thì vòng lặp sẽ tiếp tục cho đến khi số byte đọc được đúng bằng kích thước yêu cầu.

Chương trình Server gởi và nhận dữ liệu với kích thước cố định

```
    using System;
    using System.Net;
    using System.Net.Sockets;
    using System.Text;
    class FixedTcpSrvr
    {
        private static int SendData(Socket s, byte[] data)
        {
            int total = 0;
            int size = data.Length;
            int dataleft = size;
            int sent;
            while (total < size)
            {
                sent = s.Send(data, total, dataleft, SocketFlags.None);
```

```
                total += sent;
                dataleft -= sent;
        }
        return total;
}
private static byte[] ReceiveData(Socket s, int size)
{
        int total = 0;
        int dataleft = size;
        byte[] data = new byte[size];
        int recv;
        while (total < size)
        {
                recv = s.Receive(data, total, dataleft, 0);
                if (recv == 0)
                {
                        data = Encoding.ASCII.GetBytes("exit");
                        break;
                }
                total += recv;
                dataleft -= recv;
        }
        return data;
}
public static void Main()
{
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Dang cho Client ket noi den...");
        Socket client = newsock.Accept();
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
```

```
                Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
                newclient.Address, newclient.Port);
                string welcome = "Hello Client";
                data = Encoding.ASCII.GetBytes(welcome);
                int sent = SendData(client, data);
                for (int i = 0; i < 5; i++)
                {
                        data = ReceiveData(client, 12);
                        Console.WriteLine(Encoding.ASCII.GetString(data));
                }
                Console.WriteLine("Da ngat ket noi voi Client {0}", newclient.Address);
                client.Close();
                newsock.Close();
        }
}
```

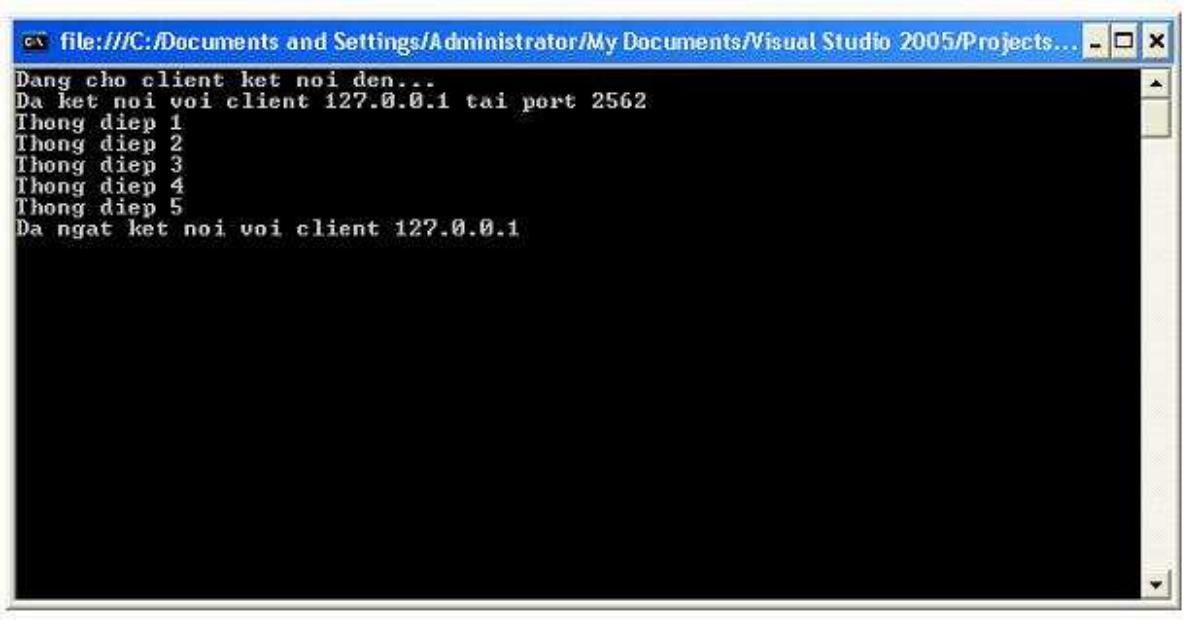
Chương trình Client gởi và nhận dữ liệu với kích thước cố định

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class FixedTcpClient
{
        private static int SendData(Socket s, byte[] data)
        {
                int total = 0;
                int size = data.Length;
                int dataleft = size;
                int sent;
                while (total < size)
                {
                        sent = s.Send(data, total, dataleft, SocketFlags.None);
                        total += sent;
                        dataleft -= sent;
                }
                return total;
```

```
}
private static byte[] ReceiveData(Socket s, int size)
{
        int total = 0;
        int dataleft = size;
        byte[] data = new byte[size];
        int recv;
        while (total < size)
        {
                recv = s.Receive(data, total, dataleft, 0);
                if (recv == 0)
                {
                        data = Encoding.ASCII.GetBytes("exit ");
                        break;
                }
                total += recv;
                dataleft -= recv;
        }
        return data;
}
public static void Main()
{
        byte[] data = new byte[1024];
        int sent;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        try
        {
                server.Connect(ipep);
        }
        catch (SocketException e)
        {
                Console.WriteLine("Khong the ket noi den server");
                Console.WriteLine(e.ToString());
```

```
                return;
            }
            int recv = server.Receive(data);
            string stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
            sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 1"));
            sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 2"));
            sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 3"));
            sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 4"));
            sent = SendData(server, Encoding.ASCII.GetBytes("Thong diep 5"));
            Console.WriteLine("Dong ket noi voi server...");
            server.Shutdown(SocketShutdown.Both);
            server.Close();
        }
    }
```

Kết quả trên Server



Hình II.7: Kết quả gởi và nhận dữ liệu với kích thước cố định

II.4.6.2. Gởi kèm kích thước thông điệp cùng với thông điệp

Cách giải quyết vấn đề biên thông điệp của TCP bằng cách sử dụng các thông điệp với kích thước cố định là một giải pháp lãng phí vì tất cả các thông điệp đều phải cùng kích thước. Nếu các thông điệp nào chưa đủ kích thước thì phải thêm phần đệm vào, gây lãng phí băng thông mạng.

Một giải pháp cho vấn đề cho phép các thông điệp được gởi với các kích thước khác nhau là gởi kích thước thông điệp kèm với thông điệp. Bằng cách này thiết bị nhận sẽ biết được kích thước của mỗi thông điệp.

Để thực hiện việc này ta sửa đổi phương thức SendData() trong ví dụ trước.

```
private static int SendVarData(Socket s, byte[] buff)
{
        int total = 0;
        int size = buff.Length;
        int dataleft = size;
        int sent;
        byte[] datasize = new byte[4];
        datasize = BitConverter.GetBytes(size);
        sent = s.Send(datasize);
        while (total < size)
        {
                sent = s.Send(buff, total, dataleft, SocketFlags.None);
                total += sent;
                dataleft -= sent;
        }
        return total;
}
```

Trong phương thức SendVarData(), ta sẽ lấy kích thước của thông điệp và gắn nó vào đầu của thông điệp, kích thước này là một số integer 4 byte. Kích thước tối đa của mỗi thông điệp này là 65KB. Giá trị integer 4 byte này đầu tiên được chuyển thành mảng các byte, hàm GetBytes() của lớp BitConverter được dùng để thực hiện việc này. Mảng kích thước sau đó được gởi đến thiết bị ở xa, sau khi gởi kích thước thông điệp xong, phần chính của thông điệp được gởi đi, kỹ thuật gởi cũng giống nhưtrong ví dụ trước, chúng ta sẽ lặp cho đến khi tất cả các byte đã được gởi.

Bước tiếp theo là tạo ra một phương thức có thể nhận 4 byte kích thước thông điệp và toàn bộ thông điệp. Phương thức ReceiveData() trong ví dụ trước được sửa đổi để thực hiện việc này.

```
private static byte[] ReceiveVarData(Socket s)
{
        int total = 0;
        int recv;
        byte[] datasize = new byte[4];
        recv = s.Receive(datasize, 0, 4, 0);
        int size = BitConverter.ToInt32(datasize, 0);
        int dataleft = size;
        byte[] data = new byte[size];
        while (total < size)
        {
```

```
                recv = s.Receive(data, total, dataleft, 0);
                if (recv == 0)
                {
                        data = Encoding.ASCII.GetBytes("exit ");
                        break;
                }
                total += recv;
                dataleft -= recv;
        }
        return data;
}
```

Hàm ReceiveVarData() nhận 4 byte đầu tiên của thông điệp và chuyển nó thành giá trị integer bằng phương thức GetInt32() của lớp BitConverter.

Chương trình Server gởi và nhận thông điệp cùng với kích thước

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class VarTcpSrvr
{
        private static int SendVarData(Socket s, byte[] data)
        {
                int total = 0;
                int size = data.Length;
                int dataleft = size;
                int sent;
                byte[] datasize = new byte[4];
                datasize = BitConverter.GetBytes(size);
                sent = s.Send(datasize);
                while (total < size)
                {
                        sent = s.Send(data, total, dataleft, SocketFlags.None);
                        total += sent;
                        dataleft -= sent;
                }
```

```
        return total;
}
private static byte[] ReceiveVarData(Socket s)
{
        int total = 0;
        int recv;
        byte[] datasize = new byte[4];
        recv = s.Receive(datasize, 0, 4, 0);
        int size = BitConverter.ToInt32(datasize, 0);
        int dataleft = size;
        byte[] data = new byte[size];
        while (total < size)
        {
                recv = s.Receive(data, total, dataleft, 0);
                if (recv == 0)
                {
                        data = Encoding.ASCII.GetBytes("exit ");
                        break;
                }
                total += recv;
                dataleft -= recv;
        }
        return data;
}
public static void Main()
{
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Dang cho Client ket noi den...");
        Socket client = newsock.Accept();
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
```

```
                Console.WriteLine("Da ket noi voi client {0} tai port {1}", newclient.Address,
                newclient.Port);
                string welcome = "Hello client";
                data = Encoding.ASCII.GetBytes(welcome);
                int sent = SendVarData(client, data);
                for (int i = 0; i < 5; i++)
                {
                        data = ReceiveVarData(client);
                        Console.WriteLine(Encoding.ASCII.GetString(data));
                }
                Console.WriteLine("Dong ket noi voi Client {0}", newclient.Address);
                client.Close();
                newsock.Close();
        }
}
```

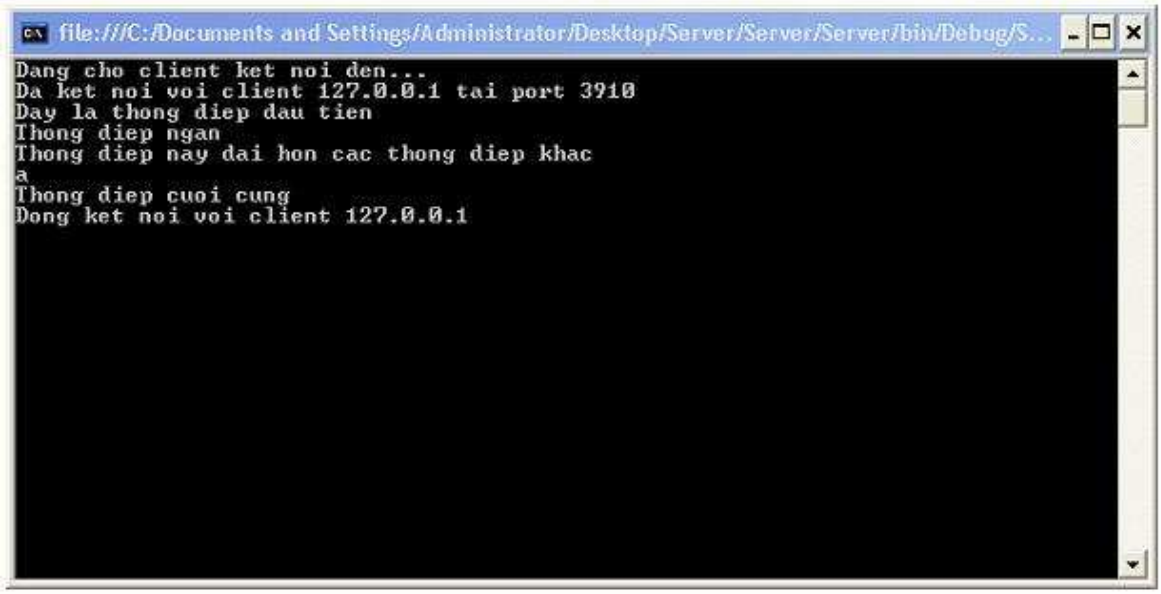
Chương trình Client gởi và nhận thông điệp cùng với kích thước

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class VarTcpClient
{
        private static int SendVarData(Socket s, byte[] data)
        {
                int total = 0;
                int size = data.Length;
                int dataleft = size;
                int sent;
                byte[] datasize = new byte[4];
                datasize = BitConverter.GetBytes(size);
                sent = s.Send(datasize);
                while (total < size)
                {
                        sent = s.Send(data, total, dataleft, SocketFlags.None);
                        total += sent;
```

```
                dataleft -= sent;
            }
            return total;
        }
        private static byte[] ReceiveVarData(Socket s)
        {
            int total = 0;
            int recv;
            byte[] datasize = new byte[4];
            recv = s.Receive(datasize, 0, 4, 0);
            int size = BitConverter.ToInt32(datasize, 0);
            int dataleft = size;
            byte[] data = new byte[size];
            while (total < size)
            {
                recv = s.Receive(data, total, dataleft, 0);
                if (recv == 0)
                {
                    data = Encoding.ASCII.GetBytes("exit ");
                    break;
                }
                total += recv;
                dataleft -= recv;
            }
            return data;
        }
        public static void Main()
        {
            byte[] data = new byte[1024];
            int sent;
            IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
            Socket server = new Socket(AddressFamily.InterNetwork,
            SocketType.Stream, ProtocolType.Tcp);
            try
            {
```

```
                server.Connect(ipep);
            }
            catch (SocketException e)
            {
                Console.WriteLine("Khong the ket noi voi server");
                Console.WriteLine(e.ToString());
                return;
            }
            data = ReceiveVarData(server);
            string stringData = Encoding.ASCII.GetString(data);
            Console.WriteLine(stringData);
            string message1 = "Day la thong diep dau tien";
            string message2 = "Thong diep ngan";
            string message3 = "Thong diep nay dai hon cac thong diep khac";
            string message4 = "a";
            string message5 = "Thong diep cuoi cung";
            sent = SendVarData(server, Encoding.ASCII.GetBytes(message1));
            sent = SendVarData(server, Encoding.ASCII.GetBytes(message2));
            sent = SendVarData(server, Encoding.ASCII.GetBytes(message3));
            sent = SendVarData(server, Encoding.ASCII.GetBytes(message4));
            sent = SendVarData(server, Encoding.ASCII.GetBytes(message5));
            Console.WriteLine("Dang ngat ket noi voi server...");
            server.Shutdown(SocketShutdown.Both);
            server.Close();
        }
}
```

Kết quả



Hình II.8: Kết quả gởi và thông điệp cùng với kích thước

II.4.6.3. Sử dụng các hệ thống đánh dấu để phân biệt các thông điệp

Một cách khác để gởi các thông điệp với kích thước khác nhau là sử dụng các hệ thống đánh dấu. Hệ thống này sẽ chia các thông điệp bởi các ký tự phân cách để báo hiệu kết thúc thông điệp.

Khi dữ liệu được nhận từ Socket, dữ liệu được kiểm tra từng ký tự một để phát hiện các ký tự phân cách, khi các ký tự phân cách được phát hiện thì dữ liệu trước ký tự phân cách chính là một thông điệp và dữ liệu sau ký tự phân cách sẽ bắt đầu một thông điệp mới.

Phương pháp này có một số hạn chế, nếu thông điệp lớn nó sẽ làm giảm tốc độ của hệ thống vì toàn bộ các ký tự của thông điệp đều phải được kiểm tra. Cũng có trường hợp một số ký tự trong thông điệp trùng với các ký tự phân cách và thông điệp này sẽ bị tách ra thành các thông điệp con, điều này làm cho chương trình chạy bị sai lệch.

II.4.7. Sử dụng C# Stream với TCP

Điều khiển thông điệp dùng giao thức TCP thường gây ra khó khăn cho các lập trình viên nên .NET Framework cung cấp một số lớp để giảm gánh nặng lập trình. Một trong những lớp đó là NetworkStream, và hai lớp dùng để gởi và nhận text sử dụng giao thức TCP là StreamWriter và StreamReader

II.4.7.1. Lớp Ị etworkStream

Lớp NetworkStream nằm trong namespace System.Net.Socket, lớp này có nhiều phương thức tạo lập để tạo một thể hiện của lớp NetworkStream nhưng phương thức tạo lập sau hay được dùng nhất:

```
Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);

NetworkStream ns = new NetworkStream(server);
```

Một số thuộc tính của lớp NetworkStream:

Thuộc Tính	Mô tả

CanRead	true nếu NetworkStream hỗ trợ đọc
CanSeek	Luôn luôn false
CanWrite	true nếu NetworkStream hỗ trợ ghi
DataAvailable	true nếu có dữ liệu để đọc

Một số phương thức của lớp NetworkStream:

Phương thức	**Mô tả**
BeginRead()	Bắt đầu đọc NetworkStream bất đồng bộ
BeginWrite()	Bắt đầu ghi NetworkStream bất đồng bộ
Close()	Đóng đối tượng NetworkStream
CreateObjRef()	Tạo ra một đối tượng dùng như là proxy cho
NetworkStreamEndRead()	Kết thúc đọc NetworkStream bất đồng bộ
EndWrite()	Kết thúc ghi NetworkStream bất đồng bộ
Equals()	So sánh hai đối tượng NetworkStreams
Flush()	Đẩy tất cả dữ liệu từ NetworkStream đi
GetHashCode()	Lấy hash code cho NetworkStream
GetLifetimeService()	Lấy đối tượng lifetime service cho NetworkStream
GetType()	Lấy kiểu NetworkStream
InitializeLifetimeService()	Lấy đối tượng lifetime service object để điều khiển chính sách lifetime cho NetworkStream
Read()	Đọc dữ liệu từ NetworkStream
ReadByte()	Đọc một byte dữ liệu từ NetworkStream
ToString()	Trả về chuỗi mô tả NetworkStream
Write()	Ghi dữ liệu từ NetworkStream

Phương thức Read() được dùng để đọc các khối dữ liệu từ NetworkStream.

Định dạng của phương thức này: int Read(byte[] buffer, int offset, int size)

Trong đó:

- buffer: mảng các byte được đọc vào
- offset: vị trí bắt đầu để đọc vào trong bộ đệm
- size: số byte tối đa đọc được

Phương thức này trả về một giá trị integer mô tả số byte thực sự đọc được từ NetworkStream và đặt dữ liệu đọc được vào buffer.

Phương thức Write() dùng để gởi các khối dữ liệu đi cũng có định dạng tương tự:

void Write(byte[] buffer, int offset, int size)

Trong đó:

- buffer: mảng các byte để ghi

- offset: vị trí bắt đầu để ghi trong bộ đệm
- size: số byte tối đa được ghi bắt đầu tại vị trí offset

Chương trình TCP Client Ị etworkStream

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class NetworkStreamTcpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        int recv;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 500);
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Khong the ket noi den server");
            Console.WriteLine(e.ToString());
            return;
        }
        NetworkStream ns = new NetworkStream(server);
        if (ns.CanRead)
        {
            recv = ns.Read(data, 0, data.Length);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        else
```

```
                {
                        Console.WriteLine("Error: Can't read from this Socket");
                        ns.Close();
                        server.Close();
                        return;
                }
                while (true)
                {
                        input = Console.ReadLine();
                        if (input == "exit")
                        break;
                        if (ns.CanWrite)
                        {
                                ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
                                ns.Flush();
                        }
                        recv = ns.Read(data, 0, data.Length);
                        stringData = Encoding.ASCII.GetString(data, 0, recv);
                        Console.WriteLine(stringData);
                }
                Console.WriteLine("Dang ngat ket noi voi server...");
                ns.Close();
                server.Shutdown(SocketShutdown.Both);
                server.Close();
        }
}
```

Chương trình này tạo ra một đối tượng NetworkStream từ đối tượng Socket:

```
NetworkStream ns = new NetworkStream(server);
```

Khi đối tượng NetworkStream được tạo ra, đối tượng Socket sẽ không được tham chiếu đến nữa cho đến khi nó bị đóng lại vào cuối chương trình, tất cả các thông tin liên lạc với Server ở xa được thực hiện thông đối tượng NetworkStream:

```
recv = ns.Read(data, 0, data.Length);
ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
ns.Flush();
```

Phương thức Flush() được dùng sau mỗi phương thức Write() để đảm bảo dữ liệu đặt vào NetworkStream sẽ lập tức được gởi đến hệ thống ở xa. Mặc dù đối tượng NetworkStream có thêm

một số chức năng của Socket nhưng vẫn còn tồn tại vấn đề với biên thông điệp. Vấn đề này được giải quyết thông qua hai lớp hỗ trợ là StreamReader và StreamWriter.

Ta có thể kiểm tra chương trình này với chương trình TCP Server đơn giản ở trên.

II.4.7.2. Lớp StreamReader và StreamWriter

Namespcace System.IO chứa hai lớp StreamReader và StreamWriter điều khiển việc đọc và ghi các thông điệp text từ mạng. Cả hai lớp đều có thể được triển khai với một đối tượng NetworkStream để xác định các hệ thống đánh dấu cho các thông điệp TCP.

Lớp StreamReader có nhiều phương thức tạo lập, trong đó phương thức tạo lập đơn giản nhất của lớp StreamReader: public StreamReader(Stream stream);

Biến stream có thể được tham chiếu đến bất kỳ kiểu đối tượng Stream nào kể cả đối tượng NetworkStream. Có nhiều phương thức và thuộc tính có thể được dùng với đối tượng StreamReader sau khi nó được tạo ra như trong bảng sau:

Phương thức	**Mô tả**
Close()	Đóng đối tượng StreamReader
CreateObjRef()	Tạo ra một đối tượng được dùng như là một proxy cho StreamReader
DiscardBufferedData()	Bỏ dữ liệu hiện tại ở StreamReader
Equals()	So sánh hai đối tượng StreamReader
GetHashCode()	Lấy hash code cho đối tượng StreamReader
GetLifetimeService()	Lấy đối tượng lifetime service object cho StreamReader
GetType()	Lấy kiểu của đối tượng StreamReader
InitializeLifetimeService()	Tạo ra một đối tượng lifetime service cho StreamReader
Peek()	Trả về byte dữ liệu hợp lệ tiếp theo từ mà không gỡ bỏ nó khỏi stream
Read()	Đọc một hoặc nhiều byte dữ liệu từ StreamReader
ReadBlock()	Đọc một nhóm các byte từ stream StreamReader và đặt nó vào một bộ đệm
ReadLine()	Đọc dữ liệu từ bắt đầu đối tượng StreamReader trở lên cho đến khi gặp ký tự xuống dòng đầu tiên
ReadToEnd()	Đọc dữ liệu cho đến khi hết stream
ToString()	Tạo ra một chuỗi mô tả đối tượng StreamReader

Tương tự đối tượng StreamReader, đối tượng StreamWriter có thể được tạo ra từ một đối tượng NetworkStream:

public StreamWriter(Stream stream);

StreamWriter cũng có nhiều phương thức và thuộc tính kết hợp với nó, một số phương thức và thuộc tính của lớp StreamReader cũng có trong đối tượng StreamWriter, ngoài ra nó còn có một số phương thức và thuộc tính riêng:

Phương thức	**Mô tả**

Flush()	Gởi tất cả dữ liệu trong bộ đệm StreamWriter ra stream
Write()	Gởi một hoặc nhiều byte dữ liệu ra stream
WriteLine()	Gởi dữ liệu cùng với ký tự xuống dòng ra stream

Phương thức ReadLine() là phương thức hay nhất của lớp StreamReader. Nó đọc các ký tự từ stream cho tới khi nó gặp ký tự xuống dòng. Tính năng này cho phép sử dụng ký tự xuống dòng như một ký tự phân tách các thông điệp. Phương thức WriteLine() của lớp StreamWriter sẽ so khớp với phương thức ReadLine của lớp StreamReader do đó việc xử lý các thông điệp TCP dễ dàng hơn.

Chương trình Stream TCP Server

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
class StreamTcpSrvr
{
    public static void Main()
    {
        string data;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Dang cho Client ket noi toi...");
        Socket client = newsock.Accept();
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
        newclient.Address, newclient.Port);
        NetworkStream ns = new NetworkStream(client);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);
        string welcome = "Hello Client";
        sw.WriteLine(welcome);
        sw.Flush();
        while (true)
        {
```

```
                try
                {
                        data = sr.ReadLine();
                }
                catch (IOException)
                {
                        break;
                }
                Console.WriteLine(data);
                sw.WriteLine(data);
                sw.Flush();
        }
        Console.WriteLine("Da dong ket noi voi Client {0}", newclient.Address);
        sw.Close();
        sr.Close();
        ns.Close();
    }
}
```

Chương trình StreamTcpSrvr dùng phương thức WriteLine() của lớp StreamWriter để gởi các thông điệp text và kết thúc bằng ký tự xuống dòng. Đối với đối tượng NetworkStream, tốt hơn hết là ta phương thức Flush() sau khi gọi phương thức WriteLine() để đảm bảo rằng tất cả dữ liệu được gởi từ bộ đệm TCP.

Điểm khác biệt của chương trình này với chương trình TCP Server đơn giản ở trên là các chương trình StreamTcpSrvr biết khi nào Client ngắt kết nối. Bởi vì phương thức ReadLine() hoạt động trên stream chứ không phải Socket nên nó không thể trả về giá trị 0 khi Client ngắt kết nối. Thay vì vậy, phương thức ReadLine() sẽ phát sinh ra một ngoại lệ nếu Client ngắt kết nối và ta phải dùng catch để bắt ngoại lệ này và xử lý khi Client ngắt kết nối:

```
try
{
    data = sr.ReadLine();
}
catch (IOException)
{
    break;
}
```

Chương trình Stream TCP Client

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
class StreamTcpClient
{
    public static void Main()
    {
        string data;
        string input;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Khong the ket noi den server");
            Console.WriteLine(e.ToString());
            return;
        }
        NetworkStream ns = new NetworkStream(server);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);
        data = sr.ReadLine();
        Console.WriteLine(data);
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
            break;
```

```
                sw.WriteLine(input);
                sw.Flush();
                data = sr.ReadLine();
                Console.WriteLine(data);
            }
            Console.WriteLine("Dang dong ket noi voi server...");
            sr.Close();
            sw.Close();
            ns.Close();
            server.Shutdown(SocketShutdown.Both);
            server.Close();
        }
}
```

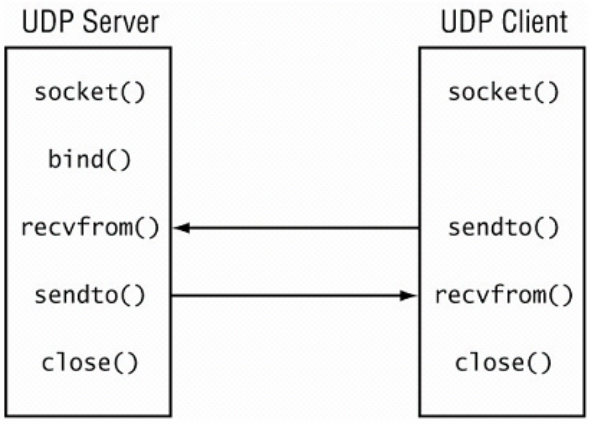
CHƯƠNG 3. LẬP TRÌNH SOCKET PHI KẾT NỐI

III.1. Tổng quan

Các Socket phi kết nối cho phép gởi các thông điệp mà không cần phải thiết lập kết nối trước. Một phương thức đọc sẽ đọc toàn bộ thông điệp được gởi bởi một phương thức gởi, điều này làm tránh được các rắc rối, phức tạp với biên dữ liệu. Thật không may mắn là giao thức phi kết nối UDP không đảm bảo dữ liệu được truyền tới đích. Nhiều yếu tố như mạng bận, mạng bị mất kết nối giữa chừng có thể ngăn cản các gói tin được truyền tới đích.

Nếu một thiết bị chờ dữ liệu từ một thiết bị ở xa, nó phải được gán một địa chỉ và port cục bộ, dùng hàm Bind() để gán. Một khi đã thực hiện xong, thiết bị có thể dùng Socket để gởi dữ liệu ra ngoài hay nhận dữ liệu từ Socket.

Bởi vì thiết bị Client không tạo ra kết nối đến một địa chỉ Server cụ thể do đó phương thức Connect() không cần dùng trong chương trình UDP Client. Mô hình bên dưới mô tả các bước lập trình Socket phi kết nối:



Hình III.1: Mô hình lập trình Socket phi kết nối

Khi kết nối không được thành lập thì phương thức Send() và Receive() không được dùng bởi vì trong hai phương thức trên đều không chỉ ra địa chỉ đích của dữ liệu. Thay vào đó, Socket phi kết nối cung cấp hai phương thức để thực hiện việc này là SendTo() và ReceiveFrom().

III.2. Lập trình phía Server

UDP là một giao thức phi kết nối do đó các lập trình viên chỉ phải làm hai việc để tạo ra một ứng dụng Server gởi và nhận dữ liệu:

- Tạo ra Socket
- Kết nối Socket đến một IPEndPoint cục bộ

Ví dụ:

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
Socket newsock = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
newsock.Bind(ipep);
```

Để thực hiện truyền thông phi kết nối, chúng ta phải chỉ ra SocketType là Dgram và ProtocolType là Udp.

Sau khi thực hiện xong hai bước trên, Socket có thể được dùng hoặc để chấp nhận các gói tin UDP đến trên IPEndPoint hoặc gởi các gói tin udp đến các thiết bị nhận khác trên mạng.

Phương thức SendTo() dùng để gởi dữ liệu, phương thức này chỉ ra dữ liệu để gởi và IPEndPoint của thiết bị nhận. Có nhiều quá tải hàm của phương thức SendTo() có thể được dùng tùy vào yêu cầu cụ thể.

SendTo(byte[] data, EndPoint Remote)

Phương thức trên gởi một mảng dữ liệu đến một EndPoint được chỉ ra bởi Remote. Một quá tải hàm khác phức tạp hơn của phương thức SendTo().

SendTo(byte[] data, SocketFlags Flags, EndPoint Remote)

Phương thức này cho phép thêm cờ SocketFlag, nó chỉ ra các tùy chọn UDP được sử dụng. Để chỉ ra số byte được gởi từ mảng byte ta sử dụng quá tải hàm sau của phương thức SendTo():

SendTo(byte[] data, int Offset, int Size, SocketFlags Flags, EndPoint Remote)

Phương thức ReceiveFrom() có dùng định dạng với phương thức SendTo(), chỉ có một điểm khác biệt sau ở cách EndPoint được khai báo. Phương thức ReceiveFrom() đơn giản được định nghĩa như sau:

ReceiveFrom(byte[] data, ref EndPoint Remote)

Cũng như thông thường, tham số thứ nhất là một mảng byte được định nghĩa để nhận dữ liệu, tham số thứ hai là truyền tham chiếu của đối tượng EndPoint. Biến này tham chiếu đến vị trí bộ nhớ nơi biến được lưu trữ. Phương thức ReceiveFrom() sẽ đặt thông tin EndPoint từ thiết bị ở xa vào vùng bộ nhớ của đối tượng EndPoint tham chiếu đến. Bằng việc sử dụng đối số thứ hai là tham chiếu ta sẽ lấy được địa chỉ IP và port của máy ở xa.

Chương trình UDP đơn giản

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleUdpSrvr
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram, ProtocolType.Udp);
        newsock.Bind(ipep);
        Console.WriteLine("Dang cho Client ket noi den...");
```

```
                IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
                EndPoint Remote = (EndPoint)(sender);
                recv = newsock.ReceiveFrom(data, ref Remote);
                Console.WriteLine("Thong diep duoc nhan tu {0}:", Remote.ToString());
                Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
                string welcome = "Hello Client";
                data = Encoding.ASCII.GetBytes(welcome);
                newsock.SendTo(data, data.Length, SocketFlags.None, Remote);
                while (true)
                {
                        data = new byte[1024];
                        recv = newsock.ReceiveFrom(data, ref Remote);
                        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
                        newsock.SendTo(data, recv, SocketFlags.None, Remote);
                }
        }
}
```

Để chương trình UDP chấp nhận các thông điệp UDP đến, nó phải được gắn với một port trên hệ thống. Việc này được thực hiện bằng cách tạo ra một đối tượng IPEndPoint sử dụng một địa chỉ IP cục bộ thích hợp, trong trường hợp này ta chỉ ra IPAddresss.Any để có thể dùng bất kỳ địa chỉ IP nào trên máy cục bộ để lắng nghe.

Sau khi gắn Socket vào một IPEndPoint, Server sẽ chờ Client kết nối đến, khi Client kết nối đến, Client sẽ gởi thông điệp đến Server. Server sau khi nhận được thông điệp từ Client nó sẽ gởi câu chào ngược lại cho Client:

```
recv = newsock.ReceiveFrom(data, ref Remote);
Console.WriteLine("Thong diep duoc nhan tu {0}:",Remote.ToString());
Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
string welcome = "Hello client";
```

Khi gởi câu chào cho Client xong, Server sẽ bắt đầu nhận và gởi thông điệp.

III.3. Lập trình phía Client

Bởi vì Client không cần chờ trên một port UDP định sắn nên nó cũng chẳng cần dùng phương thức Bind(), thay vì vậy nó sẽ lấy một port ngẫu nhien trên hệ thống khi dữ liệu được gởi và nó giữa port này để nhận dữ liệu trả về. Chương trình UDP Client cũng tương tự chương trình UDP Server:

Chương trình UDP Client đơn giản

```
using System;
using System.Net;
```

```
using System.Net.Sockets;
using System.Text;
class SimpleUdpClient
{
        public static void Main()
        {
                byte[] data = new byte[1024];
                string input, stringData;
                IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
                Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
                ProtocolType.Udp);
                string welcome = "Hello server";
                data = Encoding.ASCII.GetBytes(welcome);
                server.SendTo(data, data.Length, SocketFlags.None, ipep);
                IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
                EndPoint Remote = (EndPoint)sender;
                data = new byte[1024];
                int recv = server.ReceiveFrom(data, ref Remote);
                Console.WriteLine("Thong diep duoc nhan tu {0}:", Remote.ToString());
                Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
                while (true)
                {
                        input = Console.ReadLine();
                        if (input == "exit")
                        break;
                        server.SendTo(Encoding.ASCII.GetBytes(input), Remote);
                        data = new byte[1024];
                        recv = server.ReceiveFrom(data, ref Remote);
                        stringData = Encoding.ASCII.GetString(data, 0, recv);
                        Console.WriteLine(stringData);
                }
                Console.WriteLine("Dang dong client");
                server.Close();
        }
}
```

Chương trình UDP Client trên định nghĩa một IPEndPoint mà UDP Server sẽ gởi các gói tin:

I PEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);

Chương trình Client gởi thông điệp đến Server và chờ câu chào trả về từ Server.

Bởi vì Client không cần chấp nhận các thông điệp UDP trên một port định trước nên Client không dùng phương thức Bind(). Nó sẽ nhận các thông điệp UDP trên cùng port mà nó đã gởi.

Chương trình SimpleUdpClient đọc dữ liệu nhập vào từ bàn phím rồi gởi đến và chờ dữ liệu từ Server gởi trả về. Khi Server gởi trả dữ liệu về, Client sẽ lấy thông điệp đó ra và hiển thị lên màn hình. Nếu người dùng nhận vào "exit" thì vòng lặp sẽ thoát và kết nối bị đóng lại.

Không giống như chương trình TCP Server, chương trình UDP Server sẽ không biết khi nào Client ngắt kết nối do đó khi Client ngắt kết nối thì nó phải gởi thông điệp ngắt kết nối cho Server biết.

III.3.1. Sử dụng phương thức Connect() trong chương trình UDP Client

Các phương thức UDP được thiết kế để cho phép các lập trình viên gởi các gói tin đến bất kỳ máy nào trên mạng bất cứ lúc nào. Bởi vì giao thức UDP không yêu cầu kết nối trước khi gởi dữ liệu nên phải chỉ ra địa chỉ của máy nhận trong phương thức SendTo() và phương thức ReceiveFrom(). Nếu chương trình của chúng ta chỉ cần gởi và nhận dữ liệu từ một máy, chúng ta có thể dùng phương thức Connect().

Sau khi UDP socket được tạo ra, chúng ta có thể dùng phương thức Connect() giống như trong chương trình TCP để chỉ ra udp Server ở xa. Sau khi dùng phương thức Connect() xong ta có thể dùng phương thức Send() và Receive() để truyền tải dữ liệu giữa các thiết bị với nhau. Kỹ thuật này được minh họa trong chương trình UDP Client sau:

Chương trình udp Client dùng phương thức Connect()

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class OddUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
        Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
        ProtocolType.Udp);
        server.Connect(ipep);
        string welcome = "Xin chao server";
        data = Encoding.ASCII.GetBytes(welcome);
```

```
                server.Send(data);
                data = new byte[1024];
                int recv = server.Receive(data);
                Console.WriteLine("Nhan thong diep tu {0}:", ipep.ToString());
                Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
                while (true)
                {
                        input = Console.ReadLine();
                        if (input == "exit")
                        break;
                        server.Send(Encoding.ASCII.GetBytes(input));
                        data = new byte[1024];
                        recv = server.Receive(data);
                        stringData = Encoding.ASCII.GetString(data, 0, recv);
                        Console.WriteLine(stringData);
                }
                Console.WriteLine("Dang dong client");
                server.Close();
        }
}
```

III.3.2. Phân biệt các thông điệp UDP

Một trong những tính năng quan trọng của UDP mà TCP không có được đó là khả năng xử lý thông điệp mà không cần quan tâm đến biên thông điệp. UDP bảo vệ biên thông điệp của tất cả các thông điệp được gởi. Mỗi lần gọi phương thức ReceiveFrom() nó chỉ đọc dữ liệu được gởi từ một phương thức SendTo().

Khi UDP Socket được tạo ra, nó có thể nhận thông điệp từ bất kỳ Client nào. Để udp Socket phân biệt được Client nào gởi dữ liệu nó bắt buộc mỗi thông điệp phải được chứa trong một gói tin riêng và được đánh dấu bởi thông tin IP của thiết bị gởi. Điều này cho phép thiết bị nhận phân biệt được các thông điệp và thiết bị gởi.

Chương trình Client và Server sau sẽ minh họa điều này:

Chương trình UDP Server

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TestUdpSrvr
```

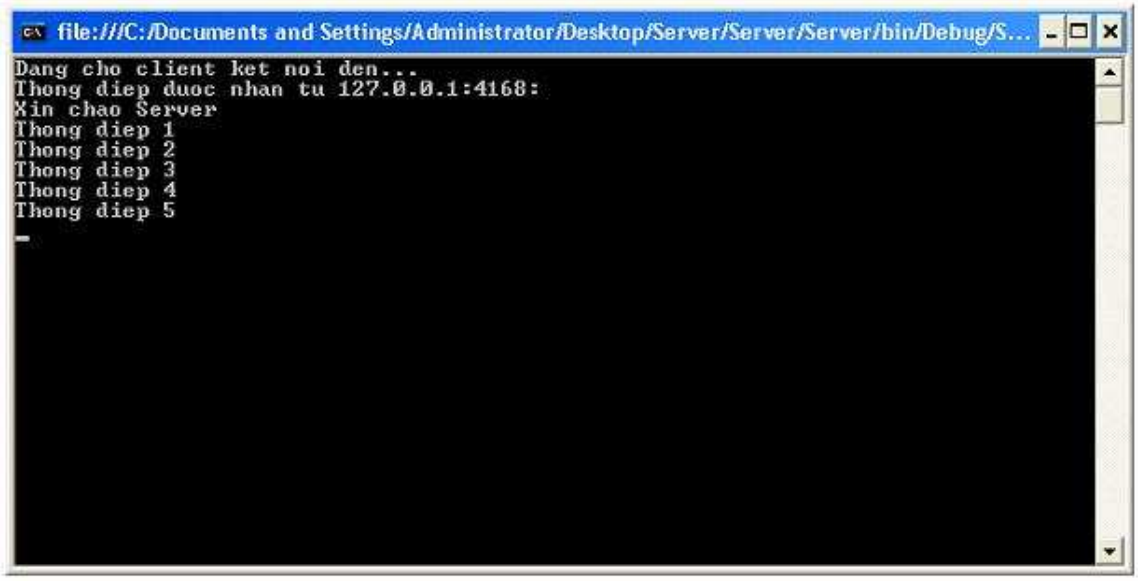
```
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram, ProtocolType.Udp);
        newsock.Bind(ipep);
        Console.WriteLine("Dang cho client ket noi den...");
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)(sender);
        recv = newsock.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
        tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        string welcome = "Xin chao client";
        data = Encoding.ASCII.GetBytes(welcome);
        newsock.SendTo(data, data.Length, SocketFlags.None, tmpRemote);
        for (int i = 0; i < 5; i++)
        {
            data = new byte[1024];
            recv = newsock.ReceiveFrom(data, ref tmpRemote);
            Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        }
        newsock.Close();
    }
}
```

Chương trình UDP Client

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TestUdpClient
{
```

```
        public static void Main()
        {
            byte[] data = new byte[1024];
            IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
            Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
            ProtocolType.Udp);
            string welcome = "Xin chao Server";
            data = Encoding.ASCII.GetBytes(welcome);
            server.SendTo(data, data.Length, SocketFlags.None, ipep);
            IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
            EndPoint tmpRemote = (EndPoint)sender;
            data = new byte[1024];
            int recv = server.ReceiveFrom(data, ref tmpRemote);
            Console.WriteLine("Thong diep duoc nhan tu {0}:",
            tmpRemote.ToString());
            Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 1"), tmpRemote);
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 2"), tmpRemote);
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 3"), tmpRemote);
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 4"), tmpRemote);
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 5"), tmpRemote);
            Console.WriteLine("Dang dong client");
            server.Close();
        }
}
```

Kết quả ở Server



Hình III.2: UDP Server nhận biết được các thông điệp riêng rẽ

III.4. Ngăn cản mất dữ liệu

Một thuận lợi của việc truyền thông dùng giao thức TCP là giao thức TCP sử dụng bộ đệm TCP. Tất cả dữ liệu được gởi bởi TCP Socket được đặt vào bộ đệm TCP trước khi được gởi ra ngoài mạng. Cũng giống như vậy, tất cả dữ liệu nhận từ Socket được đặt vào bộ đệm TCP trước khi được đọc bởi phương thức Receive(). Khi phương thức Receive() cố gắng đọc dữ liệu từ bộ đệm, nếu nó không đọc hết dữ liệu thì phần còn lại vẫn nằm trong bộ đệm và chờ lần gọi phương thức Receive() kế tiếp.

Vì UDP không quan tâm đến việc gởi lại các gói tin nên nó không dùng bộ đệm. Tất cả dữ liệu được gởi từ Socket đều được lập tức gởi ra ngoài mạng và tất cả dữ liệu được nhận từ mạng lập tức được chuyển cho phương thức ReceiveFrom() trong lần gọi tiếp theo. Khi phương thức ReceiveFrom() được dùng trong chương trình, các lập trình viên phải đảm bảo rằng bộ đệm phải đủ lớn để chấp nhận hết dữ liệu từ UDP Socket. Nếu bộ đệm quá nhỏ, dữ liệu sẽ bị mất. Để thấy được điều này, ta tiến hành thay đổi kích thước bộ đệm trong chương trình UDP đơn giản trên:

Chương trình BadUDPClient

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BadUdpClient
{
        public static void Main()
        {
                byte[] data = new byte[30];
                string input, stringData;
                IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
```

```
                Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
                ProtocolType.Udp);
                string welcome = "Xin chao server";
                data = Encoding.ASCII.GetBytes(welcome);
                server.SendTo(data, data.Length, SocketFlags.None, ipep);
                IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
                EndPoint tmpRemote = (EndPoint)sender;
                data = new byte[30];
                int recv = server.ReceiveFrom(data, ref tmpRemote);
                Console.WriteLine("Thong diep duoc nhan tu {0}:", tmpRemote.ToString());
                Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
                while (true)
                {
                        input = Console.ReadLine();
                        if (input == "exit")
                        break;
                        server.SendTo(Encoding.ASCII.GetBytes(input), tmpRemote);
                        data = new byte[30];
                        recv = server.ReceiveFrom(data, ref tmpRemote);
                        stringData = Encoding.ASCII.GetString(data, 0, recv);
                        Console.WriteLine(stringData);
                }
                Console.WriteLine("Dang dong client");
                server.Close();
        }
}
```

Ta có thể test chương trình này với chương trình UDP Server đơn giản ở trên. Khi ta nhận dữ liệu ít hơn 10 byte thì chương trình vẫn chạy bình thường nhưng khi tanhập dữ liệu lớn hơn 10 byte thì chương trình BadUdpClient sẽ phát sinh ra một biệt lệ. Mặc dầu ta không thể lấy lại dữ liệu đã bị mất nhưng ta có thể hạn chế mất dữ liệu bằng cách đặt phương thức ReceiveFrom() trong khối try-catch, khi dữ liệu bị mất bởi kích thước bộ đệm nhỏ, ta có thể tăng kích thước bộ đệm vào lần kế tiếp nhận dữ liệu.

Chương trình BetterUdpClient sau minh họa việc này:

Chương trình BetterUdpClient

```
using System;
using System.Net;
using System.Net.Sockets;
```

```
using System.Text;
class BetterdUdpClient
{
        public static void Main()
        {
                byte[] data = new byte[30];
                string input, stringData;
                IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
                Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
                ProtocolType.Udp);
                string welcome = "Xin chao server";
                data = Encoding.ASCII.GetBytes(welcome);
                server.SendTo(data, data.Length, SocketFlags.None, ipep);
                IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
                EndPoint tmpRemote = (EndPoint)sender;
                data = new byte[30];
                int recv = server.ReceiveFrom(data, ref tmpRemote);
                Console.WriteLine("Thong diep duoc nhan tu {0}:", tmpRemote.ToString());
                Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
                int i = 30;
                while (true)
                {
                        input = Console.ReadLine();
                        if (input == "exit")
                        break;
                        server.SendTo(Encoding.ASCII.GetBytes(input), tmpRemote);
                        data = new byte[i];
                        try
                        {
                                recv = server.ReceiveFrom(data, ref tmpRemote);
                                stringData = Encoding.ASCII.GetString(data, 0, recv);
                                Console.WriteLine(stringData);
                        }
                        catch (SocketException)
                        {
```

```
                    Console.WriteLine("Canh bao: du lieu bi mat, hay thu lai");
                    i += 10;
                }
            }
            Console.WriteLine("Dang dong client");
            server.Close();
        }
    }
```

Thay vì sử dụng mảng buffer với chiều dài cố định, chương trình BetterUdpClient dùng một biết có thể thiết lập giá trị khác nhau mỗi lần phương thức ReceiveFrom() được dùng.

```
data = new byte[i];
try
{
    recv = server.ReceiveFrom(data, ref tmpRemote);
    stringData = Encoding.ASCII.GetString(data, 0, recv);
    Console.WriteLine(stringData);
}
catch (SocketException)
{
    Console.WriteLine("Canh bao: du lieu bi mat, hay thu lai");
    i += 10;
}
```

III.5. Ngăn cản mất gói tin

Một khó khăn khác khi lập trình với giao thức udp là khả năng bị mất gói tin bởi vì udp là một giao thức phi kết nối nên không có cách nào mà thiết bị gởi biết được gói tin gởi có thực sự đến được đích hay không. Cách đơn giản nhất để ngăn chặn việc mất các gói tin là phải có cơ chế hồi báo giống như giao thức TCP. Các gói tin được gởi thành công đến thiết bị nhận thì thiết bị nhận phải sinh ra gói tin hồi báo cho thiết bị gởi biết đã nhận thành công. Nếu gói tin hồi báo không được nhận trong một khoảng thời gian nào đó thì thiết bị nhận sẽ cho là gói tin đó đã bị mất và gởi lại gói tin đó.

Có hai kỹ thuật dùng để truyền lại các gói tin UDP:

- Sử dụng Socket bất đồng bộ và một đối tượng Timer. Kỹ thuật này yêu cầu sử dụng một Socket bất đồng bộ mà nó có thể lắng nghe các gói tin đến không bị block. Sau khi Socket được thiết lập đọc bất đồng bộ, một đối tượng Timer có thể được thiết lập, nếu đối tượng Timer tắt trước khi hành động đọc bất đồng bộ kết thúc thì việc gởi lại dữ liệu diễn ra.
- Sử dụng Socket đồng bộ và thiết lập giá trị Socket time-out. Để làm được việc này, ta dùng phương thức SetSocketOption().

Kỹ thuật sử dụng Socket Time-out

Phương thức ReceiveFrom() là phương thức bị block. Nó sẽ block chương trình lại cho đến khi chương trình nhận dữ liệu. Nếu dữ liệu không bao giờ nhận, chương trình sẽ block mã mãi. Mặc định phương thức ReceiveFrom() sẽ bị block mãi mãi nếu không có dữ liệu được đọc. phương thức SetSocketOption() cung cấp nhiều tùy chọn cho các Socket đã được tạo, một trong những tùy chọn đó là ReceiveTimeout. Nó sẽ thiết lập khoảng thời gian Socket sẽ chờ dữ liệu đến trước khi phát ra tín hiệu time-out.

Định dạng của phương thức SetSocketOption() như sau:

SetSocketOption(SocketOptionLevel so, SocketOptionName sn, int value)

SocketOptionLevel chỉ ra kiểu tùy chọn Socket để thực thi. SocketOptionName định nghĩa tùy chọn được thiết lập, và tham số cuối cùng thiết lập giá trị cho tùy chọn.

Để chỉ ra giá trị TimeOut ta dùng như sau:

server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReceiveTimeout, 3000);

Trong đó giá trị của tham số cuối cùng chỉ ra số miligiây tối đa hàm ReceiveFrom() sẽ chờ cho đến khi có dữ liệu để đọc. Chương trình sau sẽ minh họa cách dùng TimeOut:

Chương trình TimeOutUdpClient

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TimeoutUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        int recv;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
        ProtocolType.Udp);
        int sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout mac dinh: {0}", sockopt);
        server.SetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout, 3000);
        sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout moi: {0}", sockopt);
```

```
                string welcome = "Xin chao server";
                data = Encoding.ASCII.GetBytes(welcome);
                server.SendTo(data, data.Length, SocketFlags.None, ipep);
                IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
                EndPoint tmpRemote = (EndPoint)sender;
                data = new byte[1024];
                recv = server.ReceiveFrom(data, ref tmpRemote);
                Console.WriteLine("Thong diep duoc nhan tu {0}:", mpRemote.ToString());
                Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
                while (true)
                {
                        input = Console.ReadLine();
                        if (input == "exit")
                        break;
                        server.SendTo(Encoding.ASCII.GetBytes(input), tmpRemote);
                        data = new byte[1024];
                        recv = server.ReceiveFrom(data, ref tmpRemote);
                        stringData = Encoding.ASCII.GetString(data, 0, recv);
                        Console.WriteLine(stringData);
                }
                Console.WriteLine("Dang dong client");
                server.Close();
        }
}
```

Chương trình TimeoutUdpClient đầu tiên lấy giá trị ReceiveTimeout ban đầu từ Socket và hiển thị nó, sau đó thiết lập giá trị này thành 3 giây:

```
int sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.ReceiveTimeout);
Console.WriteLine("Gia tri timeout mac dinh: {0}", sockopt);
server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReceiveTimeout,
3000);
```

Phương thức GetSocketOption() trả về một đối tượng Object, vì thế nó phải được ép kiểu thành kiểu integer. Sau khi biên dịch và chạy chương trình với chương trình SimpleUdpServer ở trên, kết quả xuất ra như sau:

```
C:\>TimeoutUdpClient
Gia tri timeout mac dinh: 0
```

Gia tri timeout moi: 3000

Unhandled Exception: System.Net.Sockets.SocketException: An existing connection was forcibly closed by the remote host

at System.Net.Sockets.Socket.ReceiveFrom(Byte[] buffer, Int32 offset, Int32 size, SocketFlags socketFlags, EndPoint& remoteEP)

at System.Net.Sockets.Socket.ReceiveFrom(Byte[] buffer, EndPoint& remoteEP)

at TimeoutUdpClient.Main()

C:\>

Giá trị ban đầu của ReceiveTimeout được thiết lập là 0 cho biết nó sẽ chờ dữ liệu mãi mãi. Sau khi thêm phương thức SetSocketOpition() và được thiết lập giá trị 3000 mili giây thì hàm ReceiveFrom() sẽ đợi dữ liệu trong 3 giây, sau 3 giây nếu không có dữ liệu để đọc thì nó sẽ phát sinh ra biệt lệ do đó ta phải đặt hàm này trong khối try – catch để xử lý biệt lệ.

III.6. Điều khiển việc truyền lại các gói tin

Có nhiều lý do các gói tin UDP không thể đến được đích, có thể lần đầu tiên gởi không tới được đích nhưng khi gởi lại lần thứ hai, ba thì tới được đích. Hầu hết các ứng dụng UDP đều cho phép gởi lại các gói tin một số lần trước khi loại bỏ nó. Khi gởi một gói tin mà không có thông điệp trả về, chúng ta có thể gởi lại thông điệp ban đầu nhiều lần, nếu sau khi gởi lại một số lần thông điệp đã đến được đích thì ta tiếp tục với phần còn lại của chương trình ngược lại ta sẽ phát sinh ra một thông báo lỗi.

Cách đơn giản nhất để thực hiện việc truyền lại là tạo ra một phương thức riêng trong lớp để điều khiển tất cả việc gởi và nhận các thông điệp. Các bước thực hiện như sau:

1) Gởi một thông điệp đến máy ở xa.

2) Chờ câu trả lời từ máy ở xa.

3) Nếu câu trả lời được nhận, chấp nhận nó và thoát khỏi phương thức với dữ liệu nhận và kích thước của dữ liệu.

4) Nếu không nhận được câu trả lời nào trong khoảng thời gian time-out, tăng biến đếm thử lên 1 đơn vị.

5) Kiểm tra biến đếm thử, nếu nó nhỏ hơn số lần đã định nghĩa trước thì quay lại bước 1, nếu nó bằng số lần đã định nghĩa trước, không truyền lại nữa và thông báo lỗi.

Một khi phương thức để gởi và nhận các gói tin udp đã được tạo ra, nó có thể được dùng bất cứ đâu trong chương trình nơi có dữ liệu được gởi tới thiết bị ở xa và chờ câu trả lời. Phương thức này được cài đặt như sau:

```
private int SndRcvData(Socket s, byte[] message, EndPoint rmtdevice)
{
        int recv;
        int retry = 0;
        while (true)
        {
                Console.WriteLine("Truyen lai lan thu: #{0}", retry);
```

```
            try
            {
                s.SendTo(message, message.Length, SocketFlags.None, rmtdevice);
                data = new byte[1024];
                recv = s.ReceiveFrom(data, ref Remote);
            }
            catch (SocketException)
            {
                recv = 0;
            }
            if (recv > 0)
            {
                return recv;
            }
            else
            {
                retry++;
                if (retry > 4)
            }
            return 0;
        }
    }
```

Phương thức này yêu cầu ba tham số:

- Một đối tượng socket đã được thành lập
- Mảng dữ liệu chứa thông điệp để gởi tới thiết bị ở xa
- Một đối tượng EndPoint chứa địa chỉ IP và port của thiết bị ở xa

Đối tượng Socket được truyền vào phương thức trên phải được khởi tạo trước và giá trị ReceiveTimeout đã được thiết lập một khoảng thời gian chờ câu trả lời từ thiết bị ở xa.

Phương thức SndRcvData() đầu tiên gởi dữ liệu đến thiết bị ở xa dùng phương thức SendTo() truyền thống. Sau khi gởi thông điệp, phương thức SndRcvData() sẽ block ở phương thức ReceiveFrom() và chờ thông điệp trả về. Nếu thông điệp được nhận từ thiết bị ở xa trong khoảng giá trị ReceiveTimeout thì phương thức SndRcvData() sẽ đặt dữ liệu vào mảng byte đã được định nghĩa trong lớp và trả về số byte đọc được. Nếu không có thông điệp trả về vào lúc kết thúc giá trị ReceiveTimeout, một biệt lệ sẽ được phát ra và khối catch được xử lý. Trong khối catch, giá trị recv được thiết lập về 0. Sau khối try-catch, giá trị recv sẽ được kiểm tra.

Nếu giá trị đó là số dương thì thông điệp đã được nhận thành công, nếu là số 0 thì không có thông điệp nào được nhận và giá trị này được tăng lên, sau đó kiểm tra nó đã đạt tới giá trị tối đa hay

chưa, nếu chưa đạt tới giá trị tối đa toàn bộ quá trình sẽ được lặp lại và bắt đầu gởi lại thông điệp, nếu đã tới giá trị tối đa rồi thì phương thức SndRcvData() sẽ trả về 0.

Chương trình RetryUdpClient

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class RetryUdpClient
{
    private byte[] data = new byte[1024];
    private static IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
    private static EndPoint Remote = (EndPoint)sender;
    private int SndRcvData(Socket s, byte[] message, EndPoint rmtdevice)
    {
        int recv;
        int retry = 0;
        while (true)
        {
            Console.WriteLine("Truyen lai lan thu: #{0}", retry);
            try
            {
                s.SendTo(message, message.Length, SocketFlags.None,
                rmtdevice);
                data = new byte[1024];
                recv = s.ReceiveFrom(data, ref Remote);
            }
            catch (SocketException)
            {
                recv = 0;
            }
            if (recv > 0)
            {
                return recv;
            }
            else
```

```
                {
                        retry++;
                        if (retry > 4)
                        {
                                return 0;
                        }
                }
        }
}
public RetryUdpClient()
{
        string input, stringData;
        int recv;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
        ProtocolType.Udp);
        int sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout mac dinh: {0}", sockopt);
        server.SetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout, 3000);
        sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,
        SocketOptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout moi: {0}", sockopt);
        string welcome = "Xin chao Server";
        data = Encoding.ASCII.GetBytes(welcome);
        recv = SndRcvData(server, data, ipep);
        if (recv > 0)
        {
                stringData = Encoding.ASCII.GetString(data, 0, recv);
                Console.WriteLine(stringData);
        }
        else
        {
                Console.WriteLine("Khong the lien lac voi thiet bi o xa");
```

```
                return;
            }
            while (true)
            {
                input = Console.ReadLine();
                if (input == "exit")
                break;
                recv = SndRcvData(server, Encoding.ASCII.GetBytes(input), ipep);
                if (recv > 0)
                {
                    stringData = Encoding.ASCII.GetString(data, 0, recv);
                    Console.WriteLine(stringData);
                }
                else
                Console.WriteLine("Khong nhan duoc cau tra loi");
            }
            Console.WriteLine("Dang dong client");
            server.Close();
        }
        public static void Main()
        {
            RetryUdpClient ruc = new RetryUdpClient();
        }
}
```

CHƯƠNG 4. SỬ DỤNG CÁC LỚP HELPER CỦA C# SOCKET

I V.1. Lớp TCP Client

Lớp TcpClient nằm ở namespace System.Net.Sockets được thiết kế để hỗ trợ cho việc viết các ứng dụng TCP Client được dễ dàng. Lớp TcpClient cho phép tạo ra một đối tượng Tcp Client, sử dụng một trong ba phương thức tạo lập như sau:

- TcpClient(): là phương thức tạo lập đầu tiên, đối tượng được tạo ra bởi phương thức tạo lập này sẽ gắn kết với một địa chỉ cục bộ và một port TCP ngẫu nhiên. Sau khi đối tượng TcpClient được tạo ra, nó phải được kết nối đến thiết bị ở xa thông qua phương thức Connect() như ví dụ dưới đây:

```
TcpClient newcon = new TcpClient();
newcon.Connect("192.168.6.1", 8000);
```

- TcpClient(IPEndPoint localEP): phương thức tạo lập này cho phép chúng ta chỉ ra địa chỉ IP cục bộ cùng với port được dùng. Đây là phương thức tạo lập thường được sử dụng khi thiết bị có nhiều hơn một card mạng và chúng ta muốn dữ liệu được gởi trên card mạng nào. Phương thức Connect() cũng được dùng để kết nối với thiết bị ở xa:

```
IPEndPoint iep = new IPEndPoint(IPAddress,Parse("192.168.6.1"), 8000);
TcpClient newcon = new TcpClient(iep);
newcon.Connect("192.168.6.2", 8000);
```

- TcpClient(String host, int port): phương thức tạo lập thứ ba này thường được sử dụng nhất, nó cho phép chỉ ra thiết bị nhận trong phương thức tạo lập và không cần phải dùng phương thức Connect(). Địa chỉ của thiết bị ở xa có thể là một chuỗi hostname hoặc một chuỗi địa chỉ IP. Phương thức tạo lập của TcpClient sẽ tự động phân giải hostname thành địa chỉ IP. Ví dụ:

```
TcpClient newcon = new TcpClient("http://www.isp.net/", 8000);
```

Mỗi khi đối tượng TcpClient được tạo ra, nhiều thuộc tính và phương thức có thể được dùng để xử lý việc truyền dữ liệu qua lại giữa các thiết bị.

Phương thức	**Mô tả**
Close()	Đóng kết nối TCP
Connect()	Thành lập kết nối TCP client với thiết bị ở xa
Equals()	So sánh hai đối tượng TcpClient
GetHashCode()	Lấy mã hash code
GetStream()	Lấy đối tượng Stream nó có thể dùng để gởi và nhận dữ liệu
GetType()	Lấy kiểu của thể hiện hiện tại
ToString()	Chuyển thể hiện hiện tại sang kiểu chuỗi

Mỗi khi kết nối được thành lập, phương thức GetStream() gán một đối tượng NetworkStream để gởi và nhận dữ liệu nhờ vào phương thức Read() và Write(). Lớp TcpClient còn có nhiều thuộc tính được mô tả trong bảng sau:

Thuộc Tính	Mô tả
LingerState	Lấy hoặc thiết lập thời gian kết nối TCP vẫn còn sau khi gọi phương thức Close()
NoDelay	Lấy hoặc thiết lập thời gian trễ được dùng để gởi hoặc nhận ở bộ đệm TCP
ReceiveBufferSize	Lấy hoặc thiết lập kích thước bộ đệm TCP nhận
ReceiveTimeout	Lấy hoặc thiết lập thời gian timeout của Socket
SendBufferSize	Lấy hoặc thiết lập kích thước bộ đệm TCP gởi
SendTimeout	Lấy hoặc thiết lập giá trị timeout của Socket

Chương trình TCPClient đơn giản

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpClientSample
{
	public static void Main()
	{
		byte[] data = new byte[1024];
		string input, stringData;
		TcpClient server;
		try
		{
			server = new TcpClient("127.0.0.1", 5000);
		}
		catch (SocketException)
		{
			Console.WriteLine("Khong the ket noi den server");
			return;
		}
		NetworkStream ns = server.GetStream();
		int recv = ns.Read(data, 0, data.Length);
		stringData = Encoding.ASCII.GetString(data, 0, recv);
```

```
                Console.WriteLine(stringData);
                while (true)
                {
                        input = Console.ReadLine();
                        if (input == "exit") break;
                        ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
                        ns.Flush();
                        data = new byte[1024];
                        recv = ns.Read(data, 0, data.Length);
                        stringData = Encoding.ASCII.GetString(data, 0, recv);
                        Console.WriteLine(stringData);
                }
                Console.WriteLine("Dang ngat ket noi voi server...");
                ns.Close();
                server.Close();
        }
}
```

Trong chương trình trên phương thức tạo lập sẽ tự động kết nối đến Server được chỉ ra ở xa, nó nên được đặt ở trong khối try-catch để phòng trường hợp Server không hợp lệ.

Sau khi đối tượng NetworkStream được tạo ra, ta có thể dùng phương thức Read() và Write() để nhận và gởi dữ liệu:

```
while (true)
{
        input = Console.ReadLine();
        if (input == "exit") break;
        ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
        ns.Flush();
        data = new byte[1024];
        recv = ns.Read(data, 0, data.Length);
        stringData = Encoding.ASCII.GetString(data, 0, recv);
        Console.WriteLine(stringData);
}
```

Phương thức Read() yêu cầu 3 tham số:

- Mảng các byte để đặt dữ liệu nhận vào
- Vị trí offset trong bộ đệm mà tại đó ta muốn đặt dữ liệu

- Chiều dài của bộ đệm dữ liệu

Cũng giống như phương thức Receive() của Socket, phương thức Read() sẽ đọc một lượng dữ liệu có độ lớn tối đa đúng bằng độ lớn bộ đệm. Nếu bộ đệm quá nhỏ, phần dữ liệu còn lại sẽ nằm ở trong stream và đợi lần gọi phương thức Read() tiếp theo.

Phương thức Write() cũng yêu cầu ba tham số:

- Mảng các byte để gởi dữ liệu
- Vị trí offset trong bộ đệm mà tại đó ta muốn gởi dữ liệu
- Chiều dài của dữ liệu được gởi

Cần chú ý rằng TCP không bảo vệ các biên thông điệp. Điều này cũng áp dụng cho lớp TcpClient, do đó ta cần phải xử lý vấn đề biên thông điệp giống như phương thức Receive() của lớp Socket bằng cách tạo ra vòng lặp để đảm bảo tất cả dữ liệu đều được đọc từ stream.

Ta có thể test chương trình này với chương trình TCP Server đơn giản ở phần trên.

IV.2. Lớp TCPListener

Cũng giống như lớp TcpClient, lớp TcpListener cũng cho phép chúng ta tạo ra các chương trình TCP Server một cách đơn giản. Lớp TcpListener có ba phương thức tạo lập:

- TcpListener(int port): gắn một đối tượng TcpListener vào một port được chỉ ra trên máy cục bộ
- TcpListener(IPEndPoint ie): gắn một đối tượng TcpListener vào một đối tượng EndPoint cục bộ
- TcpListener(IPAddress addr, int port): gắn một đối tượng TcpListener vào một đối tượng IPAddress và một port

Không giống như lớp TcpClient, các phương thức tạo lập của lớp TcpListener yêu cầu ít nhất một tham số: số port mà Server lắng nghe kết nối. Nếu Server có nhiều card mạng và ta muốn lắng nghe trên một card mạng nào đó thì ta có thể dùng một đối tượng IPEndPoint để chỉ ra địa chỉ IP của card cùng với số port dùng để lắng nghe.

Lớp TcpListener có một số phương thức sau:

Phương thức	**Mô tả**
AcceptSocket()	Chấp nhận kết nối trên port và gán kết nối cho một đối tượng Socket
AcceptTcpClient()	Chấp nhận kết nối trên port và gán kết nối cho một đối tượng TCPClient
Equals()	So sánh hai đối tượng TcpListener
GetHashCode()	Lấy hash code
GetType()	Lấy kiểu của thể hiện hiện tại
Pending()	Kiểm tra xem có yêu cầu đang chờ kết nối hay không
Start()	Bắt đầu lắng nghe kết nối
Stop()	Ngừng lắng nghe kết nối

ToString()	Chuyển đối tượng TcpListener thành chuỗi

Phương thức Start() tương tự như phương thứcBind() và Listen() được dùng ở lớp socket. Phương thức Start() kết nối Socket đến EndPoint được định nghĩa ở phương thức tạo lập của lớp TcpListener và đặt TCP port vào chế độ lắng nghe, sẵng sàng chấp nhận kết nối. Phương thức AcceptTcpClient() có thể so sánh với phương thức Accept() của Socket, chấp nhận kết nối và gán nó cho một đối tượng TcpClient.

Sau khi đối tượng TcpClient được tạo ra, tất cả các truyền thông với thiết bị ở xa được thực hiện với đối tượng TcpClient mới chứ không phải với đối tượng TcpListener ban đầu, do đó đối tượng TcpListener có thể được dùng để chấp nhận kết nối khác. Để đóng đối tượng TcpListener ta dùng phương thức Stop().

Chương trình TCPListener

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpListenerSample
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        TcpListener newsock = new TcpListener(5000);
        newsock.Start();
        Console.WriteLine("Dan cho client ket noi den...");
        TcpClient client = newsock.AcceptTcpClient();
        NetworkStream ns = client.GetStream();
        string welcome = "Hello Client";
        data = Encoding.ASCII.GetBytes(welcome);
        ns.Write(data, 0, data.Length);
        while (true)
        {
            data = new byte[1024];
            recv = ns.Read(data, 0, data.Length);
            if (recv == 0)
                break;
            Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
            ns.Write(data, 0, recv);
```

```
            }
            ns.Close();
            client.Close();
            newsock.Stop();
        }
    }
```

Chương trình TcpListenerSample đầu tiên tạo ra một đối tượng TcpListener, sử dụng port 5000 và dùng phương thức Start() để đặt đối tượng mới tạo ra vào chế độ lắng nghe. Sau đó phương thức cceptTcpClient() chờ kết nối TCP đến và gán kết nối đến này vào một đối tượng TcpClient:

```
TcpListener newsock = new TcpListener(5000);
newsock.Start();
Console.WriteLine("Dan cho client ket noi den...");
TcpClient client = newsock.AcceptTcpClient();
NetworkStream ns = client.GetStream();
```

Khi đối tượng TcpClient được thành lập, một đối tượng NetworkStream được gán vào để truyền thông với máy ở xa. Tất cả các thông tin liên lạc đều được thực hiện bằng cách sử dụng phương thức Read() và Write()

IV.3. Lớp UdpClient

Lớp UdpClient được tạo ra để giúp cho việc lập trình mạng với giao thức UDP được đơn giản hơn. Lớp UdpClient có bốn phương thức tạo lập:

- UdpClient(): tạo ra một đối tượng UdpClient nhưng không gắn vào bất kỳ địa chỉ hay port nào.
- UdpClient(int port): gắn đối tượng UdpClient mới tạo vào một port.
- UdpClient(IPEndPoint iep): gắn đối tượng UdpClient mới tạo vào một địa chỉ Ip cục bộ và một port.
- UdpClient(string host, int port): gắn đối tượng UdpClient mới tạo vào một địa chỉ IP và một port bất kỳ và kết hợp nó với một địa chỉ IP và port ở xa.

Các phương thức tạo lập của lớp UdpClient tương tự như các phương thức tạo lập của lớp TcpClient, chúng ta có thể để cho hệ thống chọn port thích hợp cho ứng dụng hoặc ta có thể chỉ ra port được dùng trong ứng dụng. Nếu ứng dụng UDP phải chấp nhận dữ liệu trên một port nào đó, ta phải định nghĩa port đó trong phương thức tạo lập của lớp UdpClient.

Một số phương thức của lớp UdpClient:

Phương thức	**Mô tả**
Close()	Đóng Socket ở bên dưới
Connect()	Cho phép chỉ ra IP endpoint ở xa để gởi và nhận dữ liệu
DropMulticastGroup()	Gỡ bỏ Socket từ 1 nhóm UDP multicast
Equals()	So sánh hai đối tượng UdpClient

GetHashCode()	Lấy hash code
GetType()	Lấy kiểu của đối tượng hiện tại
JoinMulticastGroup()	Thêm Socket vào một nhóm UDP multicast
Receive()	Nhận dữ liệu từ Socket
Send()	Gởi dữ liệu đến thiết bị ở xa từ Socket
ToString()	Chuyển đổi tượng UdpClient thành chuỗi

Có nhiều chỗ khác nhau giữa phương thức Send(), Receive() của lớp UdpClient và phương thức SendTo(), ReceiveFrom() của Socket.

Phương thức Receive()

Lớp UdpClient sử dụng phương thức Receive() để chấp nhận các gói tin trên một card mạng và một port. Chỉ có một cách sử dụng của phương thức Receive():

byte[] Receive(ref IPEndPoint iep)

Khi dữ liệu được nhận từ Socket, nó không đặt vào mảng byte trong phương thức như trong phương thức ReceiveFrom() mà nó sẽ trả về một mảng byte. Sự khác nhau thứ hai là phương thức ReceiveFrom() đặt thông tin của máy ở xa vào một đối tượng EndPoint còn phương thức Receive() đặt thông tin của máy ở xa vào một đối tượng IPEndPoint, việc này có thể làm cho lập trình viên cảm thấy dễ lập trình hơn.

Khi nhiều dữ liệu được nhận hơn kích thước bộ đệm, thay vì phát sinh ra một biệt lệ như trong phương thức ReceiveFrom() của Socket, UdpClient tả về một bộ đệm dữ liệu đủ lớn để chứa dữ liệu nhận đây là một tính năng rất hay của phương thức Receive().

Phương thức Send()

Phương thức Send() có ba quá tải hàm để gởi dữ liệu tới thiết bị ở xa:

- Send(byte[] data, int sz): gởi một mảng dữ liệu với kích thước là sz đến thiết bị ở xa mặc định. Để dùng quá tải hàm này, ta phải chỉ ra thiết bị ở xa mặc định bằng cách hoặc sử dụng phương thức tạo lập của lớp UdpClient hoặc dùng phương thức Connect().
- Send(byte[] data, int sz, IPEndPoint iep): cho phép gởi mảng dữ liệu có kích thước sz đến thiết bị ở xa được chỉ ra bởi iep.
- Send(byte[] data, int sz, string host, int port): gởi mảng dữ liệu kích thước sz đến máy ở xa và port được chỉ ra.

Chương trình UdpClient Server

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class UdpSrvrSample
{
    public static void Main()
    {
```

```
                byte[] data = new byte[1024];
                IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
                UdpClient newsock = new UdpClient(ipep);
                Console.WriteLine("Dang cho client ket noi den...");
                IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
                data = newsock.Receive(ref sender);
                Console.WriteLine("Thong diep duoc nhan tu {0}:", sender.ToString());
                Console.WriteLine(Encoding.ASCII.GetString(data, 0, data.Length));
                string welcome = "Xin chao client";
                data = Encoding.ASCII.GetBytes(welcome);
                newsock.Send(data, data.Length, sender);
                while (true)
                {
                        data = newsock.Receive(ref sender);
                        Console.WriteLine(Encoding.ASCII.GetString(data, 0, data.Length));
                        newsock.Send(data, data.Length, sender);
                }
        }
}
```

Chương trình UdpClient Client

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class UdpClientSample
{
        public static void Main()
        {
                byte[] data = new byte[1024];
                string input, stringData;
                UdpClient server = new UdpClient("127.0.0.1", 5000);
                IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
                string welcome = "Xin chao server";
                data = Encoding.ASCII.GetBytes(welcome);
                server.Send(data, data.Length);
```

```
            data = server.Receive(ref sender);
            Console.WriteLine("Thong diep duoc nhan tu {0}:", sender.ToString());
            stringData = Encoding.ASCII.GetString(data, 0, data.Length);
            Console.WriteLine(stringData);
            while (true)
            {
                input = Console.ReadLine();
                if (input == "exit")
                break;
                server.Send(Encoding.ASCII.GetBytes(input), input.Length);
                data = server.Receive(ref sender);
                stringData = Encoding.ASCII.GetString(data, 0, data.Length);
                Console.WriteLine(stringData);
            }
            Console.WriteLine("Dang dong client");
            server.Close();
        }
}
```

CHƯƠNG 5. ĐA NHIỆM TIỂU TRÌNH

V.1. Khái niệm tiến trình và tiểu trình của Windows

Tiến trình là một thể hiện của chương trình đang hoạt động. Một tiến trình luôn sở hữu một không gian địa chỉ có kích thước 4GB chứa mã chương trình, các dữ liệu, sở hữu tài nguyên của hệ thống như tập tin, đối tượng đồng bộ hóa....

Mỗi tiến trình khi mới tạo lập đều chỉ có một tiểu trình chính nhưng sau đó có thể tạo lập nhiều tiến trình khác.

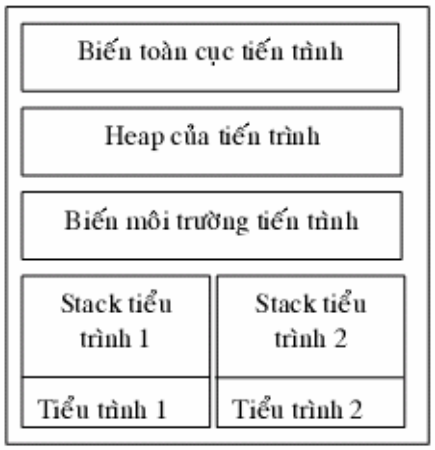
Tiểu trình là một thành phần đơn vị của tiến trình có thể thực hiện các chỉ thị ứng với một đoạn mã nào đó của chương trình.

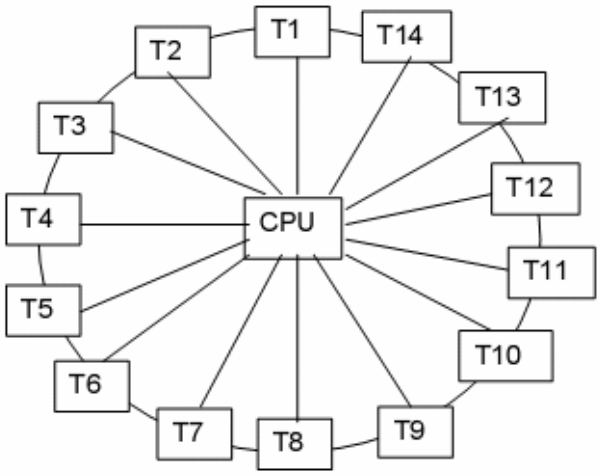
Hệ điều hành Windows cho phép các tiểu trình hoạt động độc lập và tổ chức điều phối (lập lịch tiến trình) CPU để các tiểu trình hoạt động đồng thời. Hệ điều hành phân chia thời gian sử dụng CPU cho mỗi tiến trình rất mịn theo kiểu xoay vòng.

Mỗi tiểu trình có thể có 1 trong 3 trạng thái : Running, Ready, Blocked.

Các tiểu trình trong một tiến trình có thể cùng truy xuất đến các biến toàn cục của tiến trình.

V.2. Mô hình





Các ứng dụng cài đặt theo mô hình đa tiến trình hay đa tiểu trình đều đối diện với vấn đề sau:

- Hệ thống tiêu thụ thêm bộ nhớ để lưu trữ các cấu trúc mô tả tiến trình hay tiểu trình.
- Hệ thống tốn thêm thời gian để theo vết chương trình, quản lý các tiểu trình.
- Nhiều tiến trình tranh chấp tài nguyên dùng chung đòi hỏi thực hiện đồng bộ hóa.

V.3. Các kỹ thuật trong .NET tạo tiểu trình

Thư viện lớp .NET Framework cung cấp một số phương pháp tạo tiểu trình mới:

- Thực thi một phương phức bằng tiểu trình trong Thread-pool.
- Thực thi phương thức một cách bất đồng bộ.

- Thực thi một phương thức bằng tiểu trình theo chu kỳ hay ở một thời điểm xác định.
- Thực thi phương thức bằng cách ra hiệu đối tượng WaitHandle.

V.3.1. Tạo tiểu trình trong Thread-pool

Cách tạo:

- Khai báo một phương thức chứa mã lệnh cần thực thi.
- Phương thức này phải trả về void và chỉ nhận một đối số.
- Tạo một thể hiện của ủy nhiệm System.Threading.WaitCallback tham chiếu đến phương thức này.
- Gọi phương thức tĩnh QueueUserWorkItem của lớp System.Threading.ThreadPool,
- Truyền thể hiện ủy nhiệm đã tạo làm đối số.
- Bộ thực thi sẽ xếp thể hiện ủy nhiệm này vào hàng đợi và thực thi nó khi một tiểu trình trong thread-pool sẵn sàng.

Ví dụ thực thi một phương thức có tên là DisplayMessage:

- Truyền DisplayMessage đến thread-pool hai lần
- Lần đầu không có đối số
- Lần sau có đối số là đối tượng MessageInfo

Chương trình ThreadPoolExample

```
using System;
using System.Threading;
// Lớp dùng để truyền dữ liệu cho phương thức DisplayMessage
// khi nó được thực thi bằng thread-pool.
    public class MessageInfo
    {
        private int iterations;
        private string message;
        // Phương thức khởi dựng nhận các thiết lập cấu hình cho tiểu trình.
        public MessageInfo(int iterations, string message)
        {
            this.iterations = iterations;
            this.message = message;
        }
        // Các thuộc tính dùng để lấy các thiết lập cấu hình.
        public int Iterations { get { return iterations; } }
        public string Message { get { return message; } }
    }
```

```
public class ThreadPoolExample
{
        // Hiển thị thông tin ra cửa sổ Console.
        public static void DisplayMessage(object state)
        {
                //Ép đối số state sang MessageInfo.
                MessageInfo config = state as MessageInfo;
                //Nếu đối số config = null, không có đối số nào truyền cho phương thức
                ThreadPool.QueueUserWorkItem;
                // sử dụng các giá trị mặc định.
                if (config == null)
                {
                        // Hiển thị một thông báo ra cửa sổ Console ba lần.
                        for (int count = 0; count < 3; count++)
                        {
                                Console.WriteLine("A thread-pool example.");
                                // Vào trạng thái chờ, dùng cho mục đích minh họa.
                                // Tránh đưa các tiểu trình của thread-pool
                                // vào trạng thái chờ trong các ứng dụng thực tế.
                                Thread.Sleep(1000);
                        }
                }
                else
                {
                        // Hiển thị một thông báo được chỉ định trước
                        // với số lần cũng được chỉ định trước.
                        for (int count = 0; count < config.Iterations; count++)
                        {
                                Console.WriteLine(config.Message);
                                // Vào trạng thái chờ, dùng cho mục đích minh họa.
                                // Tránh đưa các tiểu trình của thread-pool
                                // vào trạng thái chờ trong các ứng dụng thực tế.
                                Thread.Sleep(1000);
                        }
                }
```

```
        }
    public static void Main()
    {
        // Tạo một đối tượng ủy nhiệm, cho phép chúng ta
        // truyền phương thức DisplayMessage cho thread-pool.
        WaitCallback workMethod = new
        WaitCallback(ThreadPoolExample.DisplayMessage);
        // Thực thi DisplayMessage bằng thread-pool (không có đối số).
        ThreadPool.QueueUserWorkItem(workMethod);
        // Thực thi DisplayMessage bằng thread-pool (truyền một
        // đối tượng MessageInfo cho phương thức DisplayMessage).
        MessageInfo info = new MessageInfo(5, "A thread-pool example with
        arguments.");
        ThreadPool.QueueUserWorkItem(workMethod, info);
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

Tình huống sử dụng:

Khi một tiểu trình trong thread-pool sẵn sàng, nó nhận công việc kế tiếp từ hàng đợi và thực thi công việc này. Khi đã hoàn tất công việc, thay vì kết thúc, tiểu trình này quay về thread-pool và nhận công việc kế tiếp từ hàng đợi.

Việc sử dụng thread-pool giúp đơn giản hóa việc lập trình hỗ trợ đa tiểu trình. Tuy nhiên, cần lưu ý khi quyết định sử dụng thread-pool, cần xem xét các điểm sau:

- Không nên sử dụng thread-pool để thực thi các tiến trình chạy trong một thời gian dài. Vì số tiểu trình trong thread-pool là có giới hạn. Đặc biệt, tránh đặt các tiểu trình trong thread-pool vào trạng thái đợi trong một thời gian quá dài.
- Không thể điều khiển lịch trình của các tiểu trình trong thread-pool, cũng như không thể thay đổi độ ưu tiên của các công việc. Thread-pool xử lý các công việc theo thứ tự thêm chúng vào hàng đợi.

V.3.2. Tạo tiểu trình bất đồng bộ

Khi cho gọi một phương thức,thường thực hiện một cách đồng bộ; nghĩa là mã lệnh thực hiện lời gọi phải đi vào trạng thái dừng (block) cho đến khi phương thức được thực hiện xong.

Trong một số trường hợp, cần thực thi phương thức một cách bất đồng bộ; nghĩa là cho thực thi phương thức này trong một tiểu trình riêng trong khi vẫn tiếp tục thực hiện các công việc khác. Sau khi phương thức đã hoàn tất, cần lấy trị trả về của nó.

I guyên tắc hoạt động:

NET Framework hỗ trợ chế độ thực thi bất đồng bộ, cho phép thực thi bất kỳ phương thức nào một cách bất đồng bộ bằng một ủy nhiệm.

Khi khai báo và biên dịch một ủy nhiệm, trình biên dịch sẽ tự động sinh ra hai phương thức hỗ trợ chế độ thực thi bất đồng bộ: BeginInvoke và EndInvoke. Khi gọi phương thức BeginInvoke của một thể hiện ủy nhiệm, phương thức được tham chiếu bởi ủy nhiệm này được xếp vào hàng đợi để thực thi bất đồng bộ.

Quyền kiểm soát quá trình thực thi được trả về cho mã gọi BeginInvoke ngay sau đó, và phương thức được tham chiếu sẽ thực thi trong ngữ cảnh của tiểu trình sẵn sàng trước tiên trong thread-pool.

Các bước thực hiện:

- Khai báo một ủy nhiệm có chữ ký giống như phương thức cần thực thi.
- Tạo một thể hiện của ủy nhiệm tham chiếu đến phương thức này.
- Gọi phương thức BeginInvoke của thể hiện ủy nhiệm để thực thi phương thức
- Sử dụng phương thức EndInvoke để kiểm tra trạng thái của phương thức cũng như thu lấy trị trả về của nó nếu đã hoàn tất .

Các đối số của phương thức BeginInvoke gồm các đối số được chỉ định bởi ủy nhiệm, cộng với hai đối số dùng khi phương thức thực thi bất đồng bộ kết thúc:

- Một thể hiện của ủy nhiệm System.AsyncCallback tham chiếu đến phương thức mà bộ thực thi sẽ gọi khi phương thức thực thi bất đồng bộ kết thúc. Phương thức này sẽ được thực thi trong ngữ cảnh của một tiểu trình trong thread-pool. Truyền giá trị null cho đối số này nghĩa là không có phương thức nào được gọi và phải sử dụng một cơ chế khác để xác định khi nào phương thức thực thi bất bộ kết thúc.

- Một tham chiếu đối tượng mà bộ thực thi sẽ liên kết với quá trình thực thi bất đồng bộ. Phương thức thực thi bất đồng bộ không thể sử dụng hay truy xuất đến đối tượng này, nhưng mã lệnh có thể sử dụng nó khi phương thức này kết thúc, cho phép liên kết thông tin trạng thái với quá trình thực thi bất đồng bộ.

Ví dụ, đối tượng này cho phép ánh xạ các kết quả với các thao tác bất đồng bộ đã được khởi tạo trong trường hợp khởi tạo nhiều thao tác bất đồng bộ nhưng sử dụng chung một phương thức callback để xử lý việc kết thúc.

Phương thức EndInvoke cho phép lấy trị trả về của phương thức thực thi bất đồng bộ, nhưng trước hết phải xác định khi nào nó kết thúc.

Có bốn kỹ thuật dùng để xác định một phương thức thực thi bất đồng bộ đã kết thúc hay chưa:

- **Blocking:** dừng quá trình thực thi của tiểu trình hiện hành cho đến khi phương thức thực thi bất đồng bộ kết thúc. Điều này rất giống với sự thực thi đồng bộ. Tuy nhiên, nếu linh hoạt chọn thời điểm chính xác để đưa mã lệnh vào trạng thái dừng (block) thì vẫn còn cơ hội thực hiện thêm một số việc trước khi mã lệnh đi vào trạng thái này.
- **Polling:** lặp đi lặp lại việc kiểm tra trạng thái của phương thức thực thi bất đồng bộ để xác định nó kết thúc hay chưa. Đây là một kỹ thuật rất đơn giản, nhưng nếu xét về mặt xử lý thì không được hiệu quả. Nên tránh các vòng lặp chặt làm lãng phí thời gian của bộ xử lý; tốt nhất là nên đặt tiểu trình thực hiện polling vào trạng thái nghỉ (sleep) trong một khoảng thời gian bằng cách sử dụng Thread.Sleep giữa các lần kiểm tra trạng thái. Bởi kỹ thuật

polling đòi hỏi phải duy trì một vòng lặp nên hoạt động của tiểu trình chờ sẽ bị giới hạn, tuy nhiên có thể dễ dàng cập nhật tiến độ công việc

- **Waiting:** sử dụng một đối tượng dẫn xuất từ lớp System.Threading.WaitHandle để báo hiệu khi phương thức thực thi bất đồng bộ kết thúc. Waiting là một cải tiến của kỹ thuật polling, nó cho phépchờ nhiều phương thức thực thi bất đồng bộ kết thúc.

  Có thể chỉ định các giá trị time-out cho phép tiểu trình thực hiện waiting dừng lại nếu phương thức thực thi bất đồng bộ đã diễn ra quá lâu, hoặc muốn cập nhật định kỳ bộ chỉ trạng thái.

- **Callback:** Callback là một phương thức mà bộ thực thi sẽ gọi khi phương thức thực thi bất đồng bộ kết thúc. Mã lệnh thực hiện lời gọi không cần thực hiện bất kỳ thao tác kiểm tra nào, nhưng vẫn có thể tiếp tục thực hiện các công việc khác.

Callback rất linh hoạt nhưng cũng rất phức tạp, đặc biệt khi có nhiều phương thức thực thi bất đồng bộ chạy đồng thời nhưng sử dụng cùng một callback. Trong những trường hợp như thế, phải sử dụng các đối tượng trạng thái thích hợp để so trùng các phương thức đã hoàn tất với các phương thức đã khởi tạo.

Ví dụ:

Lớp AsyncExecutionExample trong mô tả cơ chế thực thi bất đồng bộ. Nó sử dụng một ủy nhiệm có tên là AsyncExampleDelegate để thực thi bất đồng bộ một phương thức có tên là LongRunningMethod.

Phương thức LongRunningMethod sử dụng Thread.Sleep để mô phỏng một phương thức có thời gian thực thi dài:

```
//Ủy nhiệm cho phép bạn thực hiện việc thực thi bất đồng bộ
//của AsyncExecutionExample.LongRunningMethod.
public delegate DateTime AsyncExampleDelegate(int delay, string name);
// Phương thức có thời gian thực thi dài.
public static DateTime LongRunningMethod(int delay, string name)
{
    Console.WriteLine("{0} : {1} example - thread starting.",
    DateTime.Now.ToString("HH:mm:ss.ffff"), name);
    // Mô phỏng việc xử lý tốn nhiều thời gian.
    Thread.Sleep(delay);
    Console.WriteLine("{0}:{1}example–thread finishing.",
    DateTime.Now.ToString("HH:mm:ss.ffff"), name);
    // Trả về thời gian hoàn tất phương thức.
    return DateTime.Now;
}
```

AsyncExecutionExample chứa 5 phương thức diễn đạt các cách tiếp cận khác nhau về việc kết thúc phương thức thực thi bất đồng bộ.

V.3.2.1. Phương thức BlockingExample

Phương thức BlockingExample thực thi bất đồng bộ phương thức LongRunningMethod và tiếp tục thực hiện công việc của nó trong một khoảng thời gian. Khi xử lý xong công việc này, BlockingExample chuyển sang trang thái dừng (block) cho đến khi phương thức LongRunningMethod kết thúc.

Để vào trạng thái dừng, BlockingExample thực thi phương thức EndInvoke của thể hiện ủy nhiệm AnsyncExampleDelegate. Nếu phương thức LongRunningMethod kết thúc, EndInvoke trả về ngay lập tức, nếu không, BlockingExample chuyển sang trạng thái dừng cho đến khi phương thức LongRunningMethod kết thúc.

Ví dụ:

```
public static void BlockingExample()
{
    Console.WriteLine(Environment.NewLine + "*** Running Blocking Example ***");
    // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null cho
    // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
    AsyncExampleDelegate longRunningMethod = new
    AsyncExampleDelegate(LongRunningMethod);
    IAsyncResult asyncResult = longRunningMethod.BeginInvoke(2000, "Blocking",
    null, null);
    // Thực hiện công việc khác cho đến khi sẵn sàng đi vào trạng thái dừng.
    for (int count = 0; count < 3; count++)
    {
        Console.WriteLine("{0} : Continue processing until " + "ready to block...",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
        Thread.Sleep(200);
    }
    // Đi vào trạng thái dừng cho đến khi phương thức
    // thực thi bất đồng bộ kết thúc và thu lấy kết quả.
    Console.WriteLine("{0} : Blocking until method is complete...",
    DateTime.Now.ToString("HH:mm:ss.ffff"));
    DateTime completion = longRunningMethod.EndInvoke(asyncResult);
    // Hiển thị thông tin kết thúc.
    Console.WriteLine("{0} : Blocking example complete.",
    completion.ToString("HH:mm:ss.ffff"));
}
```

V.3.2.2. Phương thức PollingExample

Phương thức PollingExample thực thi bất đồng bộ phương thức LongRunningMethod và sau đó thực hiện vòng lặp polling cho đến khi LongRunningMethod kết thúc.

PollingExample kiểm tra thuộc tính IsComplete của thể hiện IAsyncResult (được trả về bởi BeginInvoke) để xác định phương thức LongRunningMethod đã kết thúc hay chưa, nếu chưa, PollingExample sẽ gọi Thread.Sleep.

Ví dụ:

```
public static void PollingExample()
{
    Console.WriteLine(Environment.NewLine + " Running Polling Example");
    // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null cho
    // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
    AsyncExampleDelegate longRunningMethod = new
    AsyncExampleDelegate(LongRunningMethod);
    IAsyncResult asyncResult = longRunningMethod.BeginInvoke(2000, "Polling", null,
    null);
    // Thực hiện polling để kiểm tra phương thức thực thi
    // bất đồng bộ kết thúc hay chưa. Nếu chưa kết thúc thì đi vào
    // trạng thái chờ trong 300 mini-giây trước khi thực hiện polling lần nữa.
    Console.WriteLine("{0} : Poll repeatedly until method is complete...",
    DateTime.Now.ToString("HH:mm:ss.ffff"));
    While (!asyncResult.IsCompleted)
    {
        Console.WriteLine("{0} : Polling...",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
        Thread.Sleep(300);
    }
    // Thu lấy kết quả của phương thức thực thi bất đồng bộ.
    DateTime completion = longRunningMethod.EndInvoke(asyncResult);
    // Hiển thị thông tin kết thúc.
    Console.WriteLine("{0} : Polling example complete.",
    completion.ToString("HH:mm:ss.ffff"));
}
```

V.3.2.3. Phương thức WaitingExample

Phương thức WaitingExample thực thi bất đồng bộ phương thức LongRunningExample và sau đó chờ cho đến khi LongRunningMethod kết thúc.

WaitingExample sử dụng thuộc tính AsyncWaitHandle của thể hiện IasyncResult (được trả về bởi BeginInvoke) để có được một WaitHandle.

Gọi phương thức WaitOne của WaitHandle. Việc sử dụng giá trị time-out cho phép WaitingExample dừng quá trình đợi để thực hiện công việc khác hoặc dừng hoàn toàn nếu phương thức thực thi bất đồng bộ diễn ra quá lâu.

Ví dụ:

```
public static void WaitingExample()
{
        Console.WriteLine(Environment.NewLine + "*** Running Waiting Example ***");
        // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null cho
        // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
        AsyncExampleDelegate longRunningMethod = new
        AsyncExampleDelegate(LongRunningMethod);
        IAsyncResult asyncResult = longRunningMethod.BeginInvoke(2000, "Waiting", null,
        null);
        // Đợi phương thức thực thi bất đồng bộ kết thúc. Time-out sau 300 mili-giây và hiển
        // thị trạng thái ra cửa sổ Console trước khi tiếp tục đợi.
        Console.WriteLine("{0} : Waiting until method is complete...",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
        while (!asyncResult.AsyncWaitHandle.WaitOne(300, false))
        {
                Console.WriteLine("{0} : Wait timeout...",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
        }
        // Thu lấy kết quả của phương thức thực thi bất đồng bộ.
        DateTime completion = longRunningMethod.EndInvoke(asyncResult);
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : Waiting example complete.",
        completion.ToString("HH:mm:ss.ffff"));
}
```

V.3.2.4. Phương thức WaitAllExample

Phương thức WaitAllExample thực thi bất đồng bộ phương thức LongRunningMethod nhiều lần, sau đó sử dụng mảng các đối tượng WaitHandle để đợi đến khi tất cả các phương thức kết thúc.

```
public static void WaitAllExample()
{
        Console.WriteLine(Environment.NewLine + "*** Running WaitAll Example ***");
        // Một ArrayList chứa các thể hiện IAsyncResult cho các phương thức
        // thực thi bất đồng bộ.
        ArrayList asyncResults = new ArrayList(3);
        // Gọi ba lần LongRunningMethod một cách bất đồng bộ.
        // Truyền null cho cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
        // Thêm thể hiện IAsyncResult cho mỗi phương thức vào ArrayList.
```

```
        AsyncExampleDelegate longRunningMethod = new
        AsyncExampleDelegate(LongRunningMethod);
        asyncResults.Add(longRunningMethod.BeginInvoke(3000, "WaitAll 1", null, null));
        asyncResults.Add(longRunningMethod.BeginInvoke(2500, "WaitAll 2", null, null));
        asyncResults.Add(longRunningMethod.BeginInvoke(1500, "WaitAll 3", null, null));
        // Tạo một mảng các đối tượng WaitHandle, sẽ được sử dụng để đợi tất cả
        // các phương thức thực thi bất đồng bộ kết thúc.
        WaitHandle[] waitHandles = new WaitHandle[3];
        for (int count = 0; count < 3; count++)
        {
            waitHandles[count] =
            ((IAsyncResult)asyncResults[count]).AsyncWaitHandle;
        }
        // Đợi cả ba phương thức thực thi bất đồng bộ kết thúc. Time-out sau 300 mili-giây
        // và hiển thị trạng thái ra cửa sổ Console trước khi tiếp tục đợi.
        Console.WriteLine("{0} : Waiting until all 3 methods are complete...",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
        While (!WaitHandle.WaitAll(waitHandles, 300, false))
        {
            Console.WriteLine("{0} : WaitAll timeout...",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
         }
        // Kiểm tra kết quả của mỗi phương thức, xác định thời gian
        // phương thức cuối cùng kết thúc.
        DateTime completion = DateTime.MinValue;
        foreach (IAsyncResult result in asyncResults)
        {
            DateTime time = longRunningMethod.EndInvoke(result);
            if (time > completion) completion = time;
        }
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : WaitAll example complete.",
        completion.ToString("HH:mm:ss.ffff"));
}
```

V.3.2.5. Phương thức CallbackExample

Phương thức CallbackExample thực thi bất đồng bộ phương thức LongRunningMethod và truyền một thể hiện ủy nhiệm AsyncCallback (tham chiếu đến phương thức CallbackHandler) cho phương thức BeginInvoke.

Phương thức CallbackHandler được gọi một cách tự động cho đến khi phương thức LongRunningMethod kết thúc. Kết quả là phương thức CallbackExample vẫn tiếp tục thực hiện các công việc.

```
public static void CallbackExample()
{
    Console.WriteLine(Environment.NewLine + "*** Running Callback Example ***");
    // Gọi LongRunningMethod một cách bất đồng bộ. Truyền một
    // thể hiện ủy nhiệm AsyncCallback tham chiếu đến phương thức CallbackHandler.
    // CallbackHandler sẽ tự động được gọi khi phương thức thực thi bất đồng bộ
    // kết thúc. Truyền một tham chiếu đến thể hiện ủy nhiệm
    // AsyncExampleDelegate như một trạng thái bất đồng bộ;
    // nếu không, phương thức callback không thể truy xuất
    // thể hiện ủy nhiệm để gọi EndInvoke.
    AsyncExampleDelegate longRunningMethod = new
    AsyncExampleDelegate(LongRunningMethod);
    IAsyncResult asyncResult = longRunningMethod.BeginInvoke(2000, "Callback", new
    AsyncCallback(CallbackHandler), longRunningMethod);
    // Tiếp tục với công việc khác.
    for (int count = 0; count < 15; count++)
    {
        Console.WriteLine("{0} : Continue processing...",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
        Thread.Sleep(200);
    }
}
// Phương thức xử lý việc kết thúc bất đồng bộ bằng
callbacks.public static void CallbackHandler(IAsyncResult result)
{
    // Trích tham chiếu đến thể hiện AsyncExampleDelegate từ thể hiện IAsyncResult.
    AsyncExampleDelegate longRunningMethod =
    (AsyncExampleDelegate)result.AsyncState;
    // Thu lấy kết quả của phương thức thực thi bất đồng bộ.
    DateTime completion = longRunningMethod.EndInvoke(result);
```

```
        // Hiển thị thông tin kết thúc.
        Console.WriteLine("{0} : Callback example complete.",
        completion.ToString("HH:mm:ss.ffff"));
    }
```

V.3.3. Thực thi phương thức bằng Timer

Kỹ thuật này giúp thực thi một phương thức trong một tiểu trình riêng theo chu kỳ hay ở một thời điểm xác định. Thông thường, rất hữu ích khi thực thi một phương thức ở một thời điểm xác định hay ở những thời khoảng xác định. Ví dụ, cần sao lưu dữ liệu lúc 1:00 AM mỗi ngày hay xóa vùng đệm dữ liệu mỗi 20 phút.

Lớp Timer giúp việc định thời thực thi một phương thức trở nên dễ dàng, cho phép thực thi một phương thức được tham chiếu bởi ủy nhiệm TimerCallback ở những thời khoảng nhất định. Phương thức được tham chiếu sẽ thực thi trong ngữ cảnh của một tiểu trình trong thread-pool.

Cách thực hiện:

- Khai báo một phương thức trả về void và chỉ nhận một đối tượng làm đối số.
- Sau đó, tạo một thể hiện ủy nhiệm System.Threading.TimerCallback tham chiếu đến phương thức này.
- Tiếp theo, tạo một đối tượng System.Threading.Timer và truyền nó cho thể hiện ủy nhiệm TimerCallback cùng với một đối tượng trạng thái mà Timer sẽ truyền cho phương thức khi Timer hết hiệu lực.
- Bộ thực thi sẽ chờ cho đến khi Timer hết hiệu lực và sau đó gọi phương thức bằng một tiểu trình trong thread-pool.

Khi tạo một đối tượng Timer, cần chỉ định hai thời khoảng (thời khoảng có thể được chỉ định là các giá trị kiểu int, long, uint, hay System.TimeSpan):

Giá trị đầu tiên là thời gian trễ (tính bằng mili-giây) để phương thức được thực thi lần đầu tiên. Chỉ định giá trị 0 để thực thi phương thức ngay lập tức, và chỉ định System.Threading.Timeout.Infinite để tạo Timer ở trạng thái chưa bắt đầu (unstarted).

Giá trị thứ hai là khoảng thời gian mà Timer sẽ lặp lại việc gọi phương thức sau lần thực thi đầu tiên. Nếu chỉ định giá trị 0 hay Timeout.Infinite thì Timer chỉ thực thi phương thức một lần duy nhất (với điều kiện thời gian trễ ban đầu không phải là Timeout.Infinite). Đối số thứ hai có thể cung cấp bằng các trị kiểu int, long, uint, hay System.TimeSpan.

Sau khi tạo đối tượng Timer, cũng có thể thay đổi các thời khoảng được sử dụng bởi Timer bằng phương thức Change, nhưng không thể thay đổi phương thức sẽ được gọi.

Khi đã dùng xong Timer, nên gọi phương thức Timer.Depose để giải phóng tài nguyên hệ thống bị chiếm giữ bởi Timer. Việc hủy Timer cũng hủy luôn phương thức đã được định thời thực thi.

Ví dụ: Lớp TimerExample trình bày cách sử dụng Timer để gọi một phương thức có tên là TimerHandler. Ban đầu, Timer được cấu hình để gọi TimerHandler sau hai giây và lặp lại sau một giây. Ví dụ này cũng trình bày cách sử dụng phương thức Timer.Change để thay đổi các thời khoảng.

Chương trình TimerExample

```
using System;
```

```
using System.Threading;
public class TimerExample
        {
                // Phương thức sẽ được thực khi Timer hết hiệu lực.
                // Hiển thị một thông báo ra cửa sổ Console.
                private static void TimerHandler(object state)
                {
                        Console.WriteLine("{0} : {1}",
                        DateTime.Now.ToString("HH:mm:ss.ffff"), state);
                }
                public static void Main()
                {
                        // Tạo một thể hiện ủy nhiệm TimerCallback mới
                        // tham chiếu đến phương thức tĩnh TimerHandler.
                        // TimerHandler sẽ được gọi khi Timer hết hiệu lực.
                        TimerCallback handler = new TimerCallback(TimerHandler);
                        // Tạo một đối tượng trạng thái, đối tượng này sẽ được
                        // truyền cho phương thức TimerHandler.
                        // Trong trường hợp này, một thông báo sẽ được hiển thị.
                        string state = "Timer expired.";
                        Console.WriteLine("{0} : Creating Timer.",
                        DateTime.Now.ToString("HH:mm:ss.ffff"));
                        // Tạo một Timer, phát sinh lần đầu tiên sau hai giây
                        // và sau đó là mỗi giây.
                        using (Timer timer = new Timer(handler, state, 2000, 1000))
                        {
                                int period;
                                // Đọc thời khoảng mới từ Console cho đến khi
                                // người dùng nhập 0. Các giá trị không hợp lệ
                                // sẽ sử dụng giá trị mặc định là 0 (dừng ví dụ).
                                do
                                {
                                        try
                                        {
                                                period = Int32.Parse(Console.ReadLine());
```

```
                    }
                    catch { period = 0; }
                    // Thay đổi Timer với thời khoảng mới.
                    if (period > 0) timer.Change(0, period);
                } while (period > 0);
            }
            // Nhấn Enter để kết thúc.
            Console.WriteLine("Main method complete. Press Enter.");
            Console.ReadLine();
        }
    }
```

V.3.4. Thực thi phương thức bằng tiểu trình mới

Thực thi mã lệnh trong một tiểu trình riêng, và muốn kiểm soát hoàn toàn quá trình thực thi và trạng thái của tiểu trình đó.

Để tăng độ linh hoạt và mức độ kiểm soát khi hiện thực các ứng dụng hỗ trợ đa tiểu trình, bạn phải trực tiếp tạo và quản lý các tiểu trình. Lớp Thread cung cấp một cơ chế mà qua đó bạn có thể tạo và kiểm soát các tiểu trình.

Các bước thực hiện:

- Tạo một đối tượng ủy nhiệm ThreadStart tham chiếu đến phương thức chứa mã lệnh mà muốn dùng một tiểu trình mới để chạy nó. Giống như các ủy nhiệm khác, ThreadStart có thể tham chiếu đến một phương thức tĩnh hay phương thức của một đối tượng. Phương thức được tham chiếu phải trả về void và không có đối số.
- Tạo một đối tượng Thread, và truyền thể hiện ủy nhiệm ThreadStart cho phương thức khởi dựng của nó. Tiểu trình mới có trạng thái ban đầu là Unstarted (một thành viên thuộc kiểu liệt kê System.Threading.ThreadState).
- Gọi thực thi phương thức Start của đối tượng Thread để chuyển trạng thái của nó sang ThreadState.Running và bắt đầu thực thi phương thức được tham chiếu bởi thể hiện ủy nhiệm ThreadStart.

Vì ủy nhiệm ThreadStart khai báo không có đối số, nên không thể truyền dữ liệu trực tiếp cho phương thức được tham chiếu. Để truyền dữ liệu cho tiểu trình mới, cần phải cấu hình dữ liệu là khả truy xuất đối với mã lệnh đang chạy trong tiểu trình mới.

Cách tiếp cận thông thường là tạo một lớp đóng gói cả dữ liệu cần cho tiểu trình và phương thức được thực thi bởi tiểu trình. Khi muốn chạy một tiểu trình mới, thì tạo một đối tượng của lớp này, cấu hình trạng thái cho nó, và rồi chạy tiểu trình.

Ví dụ

```
using System;
using System.Threading;
public class ThreadExample
```

```
{
    // Các biến giữ thông tin trạng thái.
    private int iterations;
    private string message;
    private int delay;
    public ThreadExample(int iterations, string message, int delay)
    {
        this.iterations = iterations;
        this.message = message;
        this.delay = delay;
    }
    public void Start()
    {
        // Tạo một thể hiện ủy nhiệm ThreadStart tham chiếu đến DisplayMessage.
        ThreadStart method = new ThreadStart(this.DisplayMessage);
        // Tạo một đối tượng Thread và truyền thể hiện ủy nhiệm
        // ThreadStart cho phương thức khởi dựng của nó.
        Thread thread = new Thread(method);
        Console.WriteLine("{0} : Starting new thread.",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Khởi chạy tiểu trình mới.
        thread.Start();
    }
    private void DisplayMessage()
    {
        // Hiển thị thông báo ra cửa sổ Console với số lần
        // được chỉ định (iterations), nghỉ giữa mỗi thông báo
        // một khoảng thời gian được chỉ định (delay).
        for (int count = 0; count < iterations; count++)
        {
            Console.WriteLine("{0} : {1}",
            DateTime.Now.ToString("HH:mm:ss.ffff"), message);
            Thread.Sleep(delay);
        }
    }
```

```
    public static void Main()
    {
        // Tạo một đối tượng ThreadExample.
        ThreadExample example = new ThreadExample(5, "A thread example.", 500);
        // Khởi chạy đối tượng ThreadExample.
        example.Start();
        // Tiếp tục thực hiện công việc khác.
        for (int count = 0; count < 13; count++)
        {
            Console.WriteLine("{0} : Continue processing...",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
            Thread.Sleep(200);
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

V.3.5. Điều khiển quá trình thực thi của một tiểu trình

Cần nắm quyền điều khiển khi một tiểu trình chạy và dừng, có thể tạm dừng quá trình thực thi của một tiểu trình.

Các phương thức của lớp Thread: Abort, Interrupt, Resume, Start, và Suspend cung cấp một cơ chế điều khiển mức cao lên quá trình thực thi của một tiểu trình. Mỗi phương thức này trở về tiểu trình đang gọi ngay lập tức. Kết quả là phải viết mã để bắt và thụ lý các ngoại lệ có thể bị ném khi ta cố điều khiển quá trình thực thi của một Thread.

Phương Thức	**Mô tả**
Abort()	- Kết thúc một tiểu trình bằng cách ném ngoại lệ System.Threading. ThreadAbortException trong mã lệnh đang được chạy. - Mã lệnh của tiểu trình bị hủy có thể bắt ngoại lệ ThreadAbortException để thực hiện việc dọn dẹp, nhưng bộ thực thi sẽ tự động ném ngoại lệ này lần nữa để bảo đảm tiểu trình kết thúc, trừ khi ResetAbort được gọi.
Interrupt()	- Ném ngoại lệ - System.Threading.ThreadInterruptedException (trong mã lệnh đang được chạy) lúc tiểu trình đang ở trạng thái WaitSleepJoin. Điều này nghĩa là tiểu trình này đã gọi Sleep, Join hoặc đang đợi WaitHandle. - Nếu tiểu trình này không ở trạng thái WaitSleepJoin, ThreadInterruptedException sẽ bị ném sau khi tiểu trình đi vào trạng thái WaitSleepJoin.

Resume()	- Phục hồi quá trình thực thi của một tiểu trình đã bị tạm ngưng. - Việc gọi Resume trên một tiểu trình chưa bị tạm hoãn sẽ sinh ra ngoại lệ System.Threading.ThreadStateException trong tiểu trình đang gọi.
Start()	- Khởi chạy tiểu trình mới;
Suspend()	- Tạm hoãn quá trình thực thi của một tiểu trình cho đến khi phương thức Resume được gọi. - Việc tạm hoãn một tiểu trình đã bị tạm hoãn sẽ không có hiệu lực - Việc gọi Suspend trên một tiểu trình chưa khởi chạy hoặc đã kết thúc sẽ sinh ra ngoại lệ ThreadStateException trong tiểu trình đang gọi.

Ví dụ: Lớp ThreadControlExample sau đây khởi chạy một tiểu trình thứ hai, hiển thị định kỳ một thông báo ra cửa sổ Console và rồi đi vào trạng thái nghỉ (sleep). Bằng cách nhập các lệnh tại dấu nhắc lệnh, ta có thể gián đoạn, tạm hoãn, phục hồi, và hủy bỏ tiểu trình thứ hai.

Chương trình ThreadControlExample

```
using System;
using System.Threading;
public class ThreadControlExample
{
    private static void DisplayMessage()
    {
        // Lặp đi lặp lại việc hiển thị một thông báo ra cửa sổ Console.
        while (true)
        {
            try
            {
                Console.WriteLine("{0} : Second thread running. Enter
                (S)uspend, (R)esume, (I)nterrupt, or (E)xit.",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
                // Nghỉ 2 giây.
                Thread.Sleep(2000);
            }
            catch (ThreadInterruptedException)
            {
                // Tiểu trình đã bị gián đoạn. Việc bắt ngoại lệ
                // ThreadInterruptedException cho phép ví dụ này
                // thực hiện hành động phù hợp và tiếp tục thực thi.
                Console.WriteLine("{0} : Second thread interrupted.",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
```

```
                }
                catch (ThreadAbortException abortEx)
                {
                        // Đối tượng trong thuộc tính
                        // ThreadAbortException.ExceptionState được cung cấp
                        // bởi tiểu trình đã gọi Thread.Abort.
                        // Trong trường hợp này, nó chứa một chuỗi
                        // mô tả lý do của việc hủy bỏ.
                        Console.WriteLine("{0} : Second thread aborted ({1})",
                        DateTime.Now.ToString("HH:mm:ss.ffff"),
                        abortEx.ExceptionState);
                        // Mặc dù ThreadAbortException đã được thụ lý,
                        // bộ thực thi sẽ ném nó lần nữa để bảo đảmtiểu trình kết thúc
                }
        }
}
public static void Main()
{
        // Tạo một đối tượng Thread và truyền cho nó một thể hiện
        // ủy nhiệm ThreadStart tham chiếu đến DisplayMessage.
        Thread thread = new Thread(new ThreadStart(DisplayMessage));
        Console.WriteLine("{0} : Starting second thread.",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Khởi chạy tiểu trình thứ hai.
        thread.Start();
        // Lặp và xử lý lệnh do người dùng nhập.
        char command = ' ';
        do
        {
                string input = Console.ReadLine();
                if (input.Length > 0)
                        command = input.ToUpper()[0];
                else command = ' ';
                switch (command)
                {
                        case 'S':
```

```
                    // Tạm hoãn tiểu trình thứ hai.
                    Console.WriteLine("{0} : Suspending second thread.",
                    DateTime.Now.ToString("HH:mm:ss.ffff"));
                    thread.Suspend();
                    break;
                case 'R':
                    // Phục hồi tiểu trình thứ hai.
                    try
                    {
                        Console.WriteLine("{0} : Resuming second
                        thread.",
                        DateTime.Now.ToString("HH:mm:ss.ffff"));
                        thread.Resume();
                    }
                    catch (ThreadStateException)
                    {   Console.WriteLine("{0} : Thread wasn't
                        suspended.",
                        DateTime.Now.ToString("HH:mm:ss.ffff"));
                    } break;
                case 'I':
                    // Gián đoạn tiểu trình thứ hai.
                    Console.WriteLine("{0} : Interrupting second thread.",
                    DateTime.Now.ToString("HH:mm:ss.ffff"));
                    thread.Interrupt();
                    break;
                case 'E':
                    // Hủy bỏ tiểu trình thứ hai và truyền một đối tượng
                    // trạng thái cho tiểu trình đang bị hủy,
                    // trong trường hợp này là một thông báo.
                    Console.WriteLine("{0} : Aborting second thread.",
                    DateTime.Now.ToString("HH:mm:ss.ffff"));
                    thread.Abort("Terminating example.");
                    // Đợi tiểu trình thứ hai kết thúc.
                    thread.Join();
                    break;
            }
        }
```

```
            while (command != 'E');
            // Nhấn Enter để kết thúc.
            Console.WriteLine("Main method complete. Press Enter.");
            Console.ReadLine();
        }
}
```

V.3.6. Nhận biết khi nào một tiểu trình kết thúc

Để kiểm tra một tiểu trình đã kết thúc hay chưa là kiểm tra thuộc tính Thread.IsAlive. Thuộc tính này trả về true nếu tiểu trình đã được khởi chạy nhưng chưa kết thúc hay bị hủy.

Thông thường,cần một tiểu trình để đợi một tiểu trình khác hoàn tất việc xử lý của nó. Thay vì kiểm tra thuộc tính IsAlive trong một vòng lặp, có thể sử dụng phương thức Thread.Join. Phương thức này khiến tiểu trình đang gọi dừng lại (block) cho đến khi tiểu trình được tham chiếu kết thúc.

Có thể tùy chọn chỉ định một khoảng thời gian (giá trị int hay TimeSpan) mà sau khoảng thời gian này, Join sẽ hết hiệu lực và quá trình thực thi của tiểu trình đang gọi sẽ phục hồi lại. Nếu chỉ định một giá trị time-out, Join trả về true nếu tiểu trình đã kết thúc, và false nếu Join đã hết hiệu lực.

Ví dụ: Ví dụ sau thực thi một tiểu trình thứ hai và rồi gọi Join để đợi tiểu trình thứ hai kết thúc. Vì tiểu trình thứ hai mất 5 giây để thực thi, nhưng phương thức Join chỉ định giá trị time-out là 3 giây, nên Join sẽ luôn hết hiệu lực và ví dụ này sẽ hiển thị một thông báo ra cửa sổ Console.

Chương trình ThreadFinishExample

```
using System;
using System.Threading;
public class ThreadFinishExample
{
    private static void DisplayMessage()
    {
        // Hiển thị một thông báo ra cửa sổ Console 5 lần.
        for (int count = 0; count < 5; count++)
        {   Console.WriteLine("{0} : Second thread",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
            // Nghỉ 1 giây.
            Thread.Sleep(1000);
        }
    }
    public static void Main()
    {
        // Tạo một thể hiện ủy nhiệm ThreadStart tham chiếu đến DisplayMessage.
```

```
            ThreadStart method = new ThreadStart(DisplayMessage);
            // Tạo một đối tượng Thread và truyền thể hiện ủy nhiệm
            // ThreadStart cho phương thức khởi dựng của nó.
            Thread thread = new Thread(method);
            Console.WriteLine("{0} : Starting second thread.",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
            // Khởi chạy tiểu trình thứ hai.
            thread.Start();
            // Dừng cho đến khi tiểu trình thứ hai kết thúc, hoặc Join hết hiệu lực sau 3
            giây.
            if (!thread.Join(3000))
            {       Console.WriteLine("{0} : Join timed out !!",
                    DateTime.Now.ToString("HH:mm:ss.ffff"));
            }
            // Nhấn Enter để kết thúc.
            Console.WriteLine("Main method complete. Press Enter.");
            Console.ReadLine();
        }
    }
```

V.3.7. Khởi chạy một tiến trình mới

Lớp Process cung cấp một dạng biểu diễn được quản lý cho một tiến trình của hệ điều hành. Lớp Process hiện thực bốn quá tải hàm phương thức Start. Hai trong số này là các phương thức tĩnh, cho phép chỉ định tên và các đối số cho tiến trình mới.

Ví dụ, hai lệnh dưới đây đều thực thi Notepad trong một tiến trình mới:

```
// Thực thi notepad.exe, không có đối số.
Process.Start("notepad.exe");
// Thực thi notepad.exe, tên file cần mở là đối số.
Process.Start("notepad.exe", "SomeFile.txt");
```

Hai dạng khác của phương thức Start yêu cầu tạo đối tượng ProcessStartInfo được cấu hình với các chi tiết của tiến trình cần chạy.Việc sử dụng đối tượng ProcessStartInfo cung cấp một cơ chế điều khiển tốt hơn trên các hành vi và cấu hình của tiến trình mới.

Tóm tắt một vài thuộc tính thông dụng của lớp ProcessStartInfo:

- Arguments: Các đối số dùng để truyền cho tiến trình mới
- ErrorDialog :Nếu Process.Start không thể khởi chạy tiến trình đã được chỉ định, nó sẽ ném ngoại lệ System.ComponentModel.Win32Exception. Nếu ErrorDialog là true, Start sẽ hiển thị một thông báo lỗi trước khi ném ngoại lệ

- FileName : Tên của ứng dụng.
- WindowStyle :Điều khiển cách thức hiển thị của cửa sổ. Các giá trị hợp lệ bao gồm: Hidden, Maximized, Minimized, và Normal
- WorkingDirectory : Tên đầy đủ của thư mục làm việc

Khi đã hoàn tất với một đối tượng Process, bạn nên hủy nó để giải phóng các tài nguyên hệ thống, gọi phương thức Close(), Dispose().

Ví dụ sau sử dụng Process để thực thi Notepad trong một cửa sổ ở trạng thái phóng to và mở một file có tên là C:\Temp\file.txt. Sau khi tạo, ví dụ này sẽ gọi phương thức Process.WaitForExit để dừng tiểu trình đang chạy cho đến khi tiến trình kết thúc hoặc giá trị time-out hết hiệu lực.

Chương trình StartProcessExample

```
using System;
using System.Diagnostics;
public class StartProcessExample
{
    public static void Main()
    {
        // Tạo một đối tượng ProcessStartInfo và cấu hình cho nó
        // với các thông tin cần thiết để chạy tiến trình mới.
        ProcessStartInfo startInfo = new ProcessStartInfo();
        startInfo.FileName = "notepad.exe";
        startInfo.Arguments = "file.txt";
        startInfo.WorkingDirectory = "C:\\Temp";
        startInfo.WindowStyle = ProcessWindowStyle.Maximized;
        startInfo.ErrorDialog = true;
        // Tạo một đối tượng Process mới.
        using (Process process = new Process())
        {
            // Gán ProcessStartInfo vào Process.
            process.StartInfo = startInfo;
            try
            {   // Khởi chạy tiến trình mới.
                process.Start();
                // Đợi tiến trình mới kết thúc trước khi thoát.
                Console.WriteLine("Waiting 30 seconds for process to finish.");
                process.WaitForExit(30000);
            }
```

```
                catch (Exception ex)
                {       Console.WriteLine("Could not start process.");
                        Console.WriteLine(ex);
                }
        }
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}
```

V.3.8. Kết thúc một tiến trình

Nếu khởi chạy một tiến trình mới từ mã lệnh được quản lý bằng lớp Process, có thể kết thúc tiến trình mới bằng đối tượng Process mô tả tiến trình này. Một khi đã có đối tượng Process mô tả tiến trình cần kết thúc, cần gọi phương thức CloseMainWindow hay phương thức Kill().

Phương thức CloseMainWindow gửi một thông điệp đến cửa sổ chính của ứng dụng. CloseMainWindow sẽ không kết thúc các ứng dụng không có cửa sổ chính hoặc các ứng dụng có cửa sổ chính bị vô hiệu. Với những tình huống như thế, CloseMainWindow sẽ trả về false. CloseMainWindow trả về true nếu thông điệp được gửi thành công, nhưng không bảo đảm tiến trình thật sự kết thúc.

Phương thức Kill() kết thúc một tiến trình ngay lập tức; người dùng không có cơ hội dừng việc kết thúc, và tất cả các dữ liệu chưa được lưu sẽ bị mất.

Ví dụ sau, khởi chạy một thể hiện mới của Notepad, đợi 5 giây, sau đó kết thúc tiến trình Notepad. Trước tiên, ví dụ này kết thúc tiến trình bằng CloseMainWindow. Nếu CloseMainWindow trả về false, hoặc tiến trình Notepad vẫn cứ chạy sau khi CloseMainWindow được gọi, ví dụ này sẽ gọi Kill() và buộc tiến trình Notepad kết thúc. Có thể buộc CloseMainWindow trả về false bằng cách bỏ mặc hộp thoại File Open mở.

Chương trình TerminateProcessExample

```
using System;
using System.Threading;
using System.Diagnostics;
public class TerminateProcessExample
{
    public static void Main()
    {
        // Tạo một Process mới và chạy notepad.exe.
        using (Process process = Process.Start("notepad.exe"))
        {
```

```
                    // Đợi 5 giây và kết thúc tiến trình Notepad.
                    Thread.Sleep(5000);
                    // Kết thúc tiến trình Notepad. Gửi một thông điệp đến cửa sổ chính.
                    if (!process.CloseMainWindow())
                    {
                        // Không gửi được thông điệp. Kết thúc Notepad bằng Kill.
                        process.Kill();
                    }
                    else
                    {
                        // Thông điệp được gửi thành công; đợi 2 giây
                        // để chứng thực việc kết thúc trước khi viện đến Kill.
                        if (!process.WaitForExit(2000))
                        {
                            process.Kill();
                        }
                        // Nhấn Enter để kết thúc.
                        Console.WriteLine("Main method complete. Press Enter.");
                        Console.ReadLine();
                    }
                }
            }
        }
```

V.4. Thực thi phương thức bằng cách ra hiệu đối tượng WaitHandle

Sử dụng các lớp dẫn xuất từ WaitHandle để gọi thực thi một phương thức. Bằng phương thức RegisterWaitForSingleObject của lớp ThreadPool, có thể đăng ký thể hiện ủy nhiệm WaitOrTimerCallback với thread-pool khi một đối tượng dẫn xuất từ WaitHandle đi vào trạng thái signaled.

Có thể cấu hình thread-pool để thực thi phương thức chỉ một lần hay tự động đăng ký lại phương thức mỗi khi WaitHandle đi vào trạng thái signaled. Nếu WaitHandle đã ở trạng thái signaled khi gọi RegisterWaitForSingleObject, phương thức sẽ thực thi ngay lập tức. Phương thức Unregister của đối tượng System.Threading.RegisteredWaitHandle (được trả về bởi phương thức RegisterWaitForSingleObject) được sử dụng để hủy bỏ việc đăng ký.

Lớp thường được dùng làm bộ kích hoạt là AutoResetEvent, nó sẽ tự động chuyển sang trạng thái unsignaled sau khi ở trạng thái signaled. Tuy nhiên, cũng có thể thay đổi trạng thái signaled theo ý muốn bằng lớp ManualResetEvent hay Mutex.

CHƯƠNG 6. ĐỒNG BỘ HÓA

VI.1. Lý do đồng bộ hóa

Trong các hệ điều hành đa nhiệm cho phép nhiều công việc được thực hiện đồng thời. Việc tồn tại cùng lúc nhiều tiểu trình trong môi trường có thể dẫn đến sự tranh chấp, ngăn cản hoạt động lẫn nhau giữa các tiểu trình.

Ví dụ, với một tiến trình mới đầu tạo lập 4 tiểu trình cùng có nội dung xử lý đồng nhất: Mỗi tiểu trình sẽ tăng giá trị của biến toàn cục Count lên 250.000 lần. Do vậy Count sẽ tăng lên 1.000.000 lần trong tòan bộ tiến trình.

Để hạn chế các trình trạng tranh chấp tài nguyên cần có cơ chế điều khiển các tiểu trình truy xuất tài nguyên một cách tuần tự, đó chính là thực hiện đồng bộ hóa các tiểu trình.

Các trường hợp cần thực hiện đồng bộ hóa.

Khi một tiểu trình truy xuất đến một tài nguyên dùng chung, cần chú ý thực hiện đồng bộ hóa việc truy xuất tài nguyên để tránh xảy ra tranh chấp. Các cơ chế đồng bộ hóa đều dựa trên ý tưởng chỉ cho phép một tiểu trình được truy xuất tài nguyên khi thỏa điều kiện không có tranh chấp trên đó. Những tiểu trình không hội đủ điều kiện để sử dụng tài nguyên thì được hệ điều hành đặt vào trạng thái chờ (Không chiếm CPU), và hệ điều hành sẽ luôn kiểm sóat tình trạng truy xuất tài nguyên của tiểu trình khác để có thể giải phóng kịp thời các tiểu trình đang chờ vào thời điểm thích hợp.

VI.2. Các phương pháp đồng bộ hóa

Để thực hiện được cơ chế đồng bộ hóa , hệ điều hành sử dụng các đối tượng đồng bộ hóa gắn liền với các tài nguyên để phản ánh tình trạng truy xuất trên các tài nguyên.

Các đối tượng đồng bộ hóa có thể nhận trạng thái TRUE hoặc FALSE. Khi đối tượng đồng bộ hóa ở trạng thái TRUE tiểu trình được phép truy cập tài nguyên, ngược lại thì không.

VI.3. Phương pháp Semaphore

Khái niệm: Semaphore là một đối tượng đồng bộ hóa lưu trữ một biến đếm có giá trị từ 0 đến Max, semaphore nhận trạng thái TRUE khi giá trị của biến đếm > 0 và nhận trạng thái FALSE nếu có giá trị biến đếm = 0.

Tình huống sử dụng: Sử dụng semaphore để kiểm soát việc cho phép một số hữu hạn tiểu trình cùng lúc truy xuất một tài nguyên dùng chung.

Biến đếm của đối tượng semaphore sẽ cho biết số tiểu trình đang truy xuất tài nguyên mà semaphore bảo vệ, biến đếm sẽ giảm 1 nếu có thêm một tiểu trình truy xuất tài nguyên, ngược lại sẽ tăng giá trị lên 1 nếu có một tiểu trình chấm dứt truy xuất tài nguyên. Khi biến đếm đạt giá trị 0 tài nguyên được bảo vệ, không tiểu trình nào ngòai Max tiểu trình đã đã đăng ký được truy xuất. Có thể dùng semaphore để đồng bộ hóa các tiểu trình trong cùng hoặc khác tiến trình.

Cách tạo đối tượng Semaphore

Class Semaphore: Semaphore(int InitCount, int MaxCount)

Đăng ký truy cập tài nguyên chung: Phương thức WaitOne( )

Kết thúc truy cập tài nguyên chung: Phương thức Release( )

Ví dụ: Tạo ra 10 tiểu trình, tại một thời điểm chỉ có 3 tiểu trình truy cập tài nguyên chung:

```
class SemaphoreTest
{
        static Semaphore s = new Semaphore(3, 3);
        // Available=3; Capacity=3
        static void Main()
        {
                for (int i = 0; i < 10; i++)
                {
                        Thead thread = new Thread(new ThreadStart(Go));
                        thread.Start();
                }
        }
        static void Go()
        {
                while (true)
                {
                        s.WaitOne();
                        Thread.Sleep(100);
                        // Only 3 threads can get here at once
                        s.Release();
                }
        }
}
```

VI.4. Phương pháp dùng lớp Monitor

Một cơ chế đồng bộ hóa khác là lớp Monitor. Lớp này cho phép một tiểu trình đơn thu lấy khóa (lock) trên một đối tượng bằng cách gọi phương thức tĩnh Monitor.Enter. Bằng cách thu lấy khóa trước khi truy xuất một tài nguyên hay dữ liệu dùng chung, ta chắc chắn rằng chỉ có một tiểu trình có thể truy xuất tài nguyên đó cùng lúc. Một khi đã hoàn tất với tài nguyên, tiểu trình này sẽ giải phóng khóa để tiểu trình khác có thể truy xuất nó bằng phương thức tĩnh Monitor.Exit.

Khối mã được gói trong lệnh lock tương đương với gọi Monitor.Enter khi đi vào khối mã này, và gọi Monitor.Exit khi đi ra khối mã này. Tiểu trình chủ có thể gọi Monitor.Wait để giải phóng lock và đặt tiểu trình này vào hàng chờ (wait queue).

Các tiểu trình trong hàng chờ cũng có trạng thái là WaitSleepJoin và sẽ tiếp tục block cho đến khi tiểu trình chủ gọi phương thức Pulse hay PulseAll của lớp Monitor.

Phương thức Pulse di chuyển một trong các tiểu trình từ hàng chờ vào hàng sẵn sàng, còn phương thức PulseAll thì di chuyển tất cả các tiểu trình. Khi một tiểu trình đã được di chuyển từ hàng chờ vào hàng sẵn sàng, nó có thể thu lấy lock trong lần giải phóng kế tiếp.

Lớp ThreadSyncExample trình bày cách sử dụng lớp Monitor và lệnh lock.

Ví dụ này, khởi chạy ba tiểu trình, mỗi tiểu trình (lần lượt) thu lấy lock của một đối tượng có tên là consoleGate. Kế đó, mỗi tiểu trình gọi phương thức Monitor.Wait. Khi người dùng nhấn Enter lần đầu tiên, Monitor.Pulse sẽ được gọi để giải phóng một tiểu trình đang chờ. Lần thứ hai người dùng nhấn Enter, Monitor.PulseAll sẽ được gọi để giải phóng tất cả các tiểu trình đang chờ còn lại.

Chương trình ThreadSyncExample

```
using System;
using System.Threading;
public class ThreadSyncExample
{
    private static object consoleGate = new Object();
    private static void DisplayMessage()
    {
        Console.WriteLine("{0} : Thread started, acquiring lock...",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
        // Thu lấy chốt trên đối tượng consoleGate.
        try
        {
            Monitor.Enter(consoleGate);
            Console.WriteLine("{0} : {1}",
            DateTime.Now.ToString("HH:mm:ss.ffff"), "Acquired consoleGate
            lock, waiting...");
            // Đợi cho đến khi Pulse được gọi trên đối tượng consoleGate.
            Monitor.Wait(consoleGate);
            Console.WriteLine("{0} : Thread pulsed, terminating.",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
        }
        finally
        {
            Monitor.Exit(consoleGate);
        }
    }
    public static void Main()
    {
```

```
            // Thu lấy chốt trên đối tượng consoleGate.
            lock (consoleGate)
            {
                // Tạo và khởi chạy ba tiểu trình mới
                // (chạy phương thức DisplayMesssage).
                for (int count = 0; count < 3; count++)
                {
                    (new Thread(new ThreadStart(DisplayMessage))).Start();
                }
            }
            Thread.Sleep(1000);
            // Đánh thức một tiểu trình đang chờ.
            Console.WriteLine("{0} : {1}", DateTime.Now.ToString("HH:mm:ss.ffff"),
            "Press Enter to pulse one waiting thread.");
            Console.ReadLine();
            // Thu lấy chốt trên đối tượng consoleGate.
            lock (consoleGate)
            {
                // Pulse một tiểu trình đang chờ.
                Monitor.Pulse(consoleGate);
            }
            // Đánh thức tất cả các tiểu trình đang chờ.
            Console.WriteLine("{0} : {1}", DateTime.Now.ToString("HH:mm:ss.ffff"),
            "Press Enter to pulse all waiting threads.");
            Console.ReadLine();
            // Thu lấy chốt trên đối tượng consoleGate.
            lock (consoleGate)
            {       // Pulse tất cả các tiểu trình đang chờ.
                Monitor.PulseAll(consoleGate);
            }
            // Nhấn Enter để kết thúc.
            Console.WriteLine("Main method complete. Press Enter.");
            Console.ReadLine();
        }
}
```

VI.5. System.Threading.WaitHandle, bao gồm AutoResetEvent, ManualResetEvent

Các lớp thông dụng khác dùng để đồng bộ hóa tiểu trình là các lớp con của lớp System.Threading.WaitHandle, bao gồm AutoResetEvent, ManualResetEvent. Thể hiện của các lớp này có thể ở trạng thái signaled hay unsignaled.

Các tiểu trình có thể sử dụng các phương thức của các lớp được liệt kê để đi vào trạng thái WaitSleepJoin và đợi trạng thái của một hay nhiều đối tượng dẫn xuất từ WaitHandle biến thành signaled.

Phương thức	Mô tả
WaitAny()	Tiểu trình gọi phương thức tĩnh này sẽ đi vào trạng thái WaitSleepJoin và đợi bất kỳ một trong các đối tượng WaitHandle thuộc một mảng WaitHandle biến thành signaled. Cũng có thể chỉ định giá trị time-out.
WaitAll()	Tiểu trình gọi phương thức tĩnh này sẽ đi vào trạng thái WaitSleepJoin và đợi tất cả các đối tượng WaitHandle trong một mảng WaitHandle biến thành signaled. Bạn cũng có thể chỉ định giá trị time-out.
WaitOne()	Tiểu trình gọi phương thức này sẽ đi vào trạng thái WaitSleepJoin và đợi một đối tượng WaitHandle cụ thể biến thành signaled.

Điểm khác biệt chính giữa các lớp AutoResetEvent, ManualResetEvent, là cách thức chúng chuyển trạng thái từ signaled thành unsignaled.

Lớp AutoResetEvent và ManualResetEvent là cục bộ đối với một tiến trình. Để ra hiệu một AutoResetEvent, bạn hãy gọi phương thức Set của nó, phương thức này chỉ giải phóng một tiểu trình đang đợi sự kiện. AutoResetEvent sẽ tự động trở về trạng thái unsignaled.

Lớp ManualResetEvent phải được chuyển đổi qua lại giữa signaled và unsignaled bằng phương thức Set và Reset của nó. Gọi Set trên một ManualResetEvent sẽ đặt trạng thái của nó là signaled, giải phóng tất cả các tiểu trình đang đợi sự kiện. Chỉ khi gọi Reset mới làm cho ManualResetEvent trở thành unsignaled.

Sử dụng các lớp dẫn xuất từ WaitHandle để gọi thực thi một phương thức. Bằng phương thức RegisterWaitForSingleObject của lớp ThreadPool, có thể đăng ký thể hiện ủy nhiệm WaitOrTimerCallback với thread-pool khi một đối tượng dẫn xuất từ WaitHandle đi vào trạng thái signaled.

Có thể cấu hình thread-pool để thực thi phương thức chỉ một lần hay tự động đăng ký lại phương thức mỗi khi WaitHandle đi vào trạng thái signaled. Nếu WaitHandle đã ở trạng thái signaled khi gọi RegisterWaitForSingleObject, phương thức sẽ thực thi ngay lập tức.

Phương thức Unregister của đối tượng System.Threading.RegisteredWaitHandle (được trả về bởi phương thức RegisterWaitForSingleObject) được sử dụng để hủy bỏ việc đăng ký.

Lớp thường được dùng làm bộ kích hoạt là AutoResetEvent, nó sẽ tự động chuyển sang trạng thái unsignaled sau khi ở trạng thái signaled. Tuy nhiên, chúng ta cũng có thể thay đổi trạng thái signaled theo ý muốn bằng lớp ManualResetEvent hay Mutex.

Ví dụ dưới đây trình bày cách sử dụng một AutoResetEvent để kích hoạt thực thi một phương thức có tên là EventHandler:

Chương trình EventExecutionExample

```
using System.Threading;
```

```
public class EventExecutionExample
{
    // Phương thức sẽ được thực thi khi AutoResetEvent đi vào trạng
    // thái signaled hoặc quá trình đợi hết thời gian (time-out).
    private static void EventHandler(object state, bool timedout)
    {
        // Hiển thị thông báo thích hợp ra cửa sổ Console tùy vào quá trình đợi đã hết
        thời gian hay AutoResetEvent đã ở trạng thái signaled.
        if (timedout)
        {
            Console.WriteLine("{0} : Wait timed out.",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
        }
        else
        {
            Console.WriteLine("{0} : {1}",
            DateTime.Now.ToString("HH:mm:ss.ffff"), state);
        }
    }
    public static void Main()
    {
    // Tạo một AutoResetEvent ở trạng thái unsignaled.
    AutoResetEvent[] autoEvent;
    autoEvent[0] = new AutoResetEvent(false);
    // Tạo một thể hiện ủy nhiệm WaitOrTimerCallback tham chiếu đến
    // phương thức tĩnh EventHandler. EventHandler sẽ được gọi khi AutoResetEvent đi
    // vào trạng thái signaled hay quá trình đợi hết thời gian.
    WaitOrTimerCallback handler = new WaitOrTimerCallback(EventHandler);
    // Tạo đối tượng trạng thái (được truyền cho phương thức thụ lý sự kiện khi nó được
    // kích hoạt). Trong trường hợp này, một thông báo sẽ được hiển thị.
    string state = "AutoResetEvent signaled.";
    // Đăng ký thể hiện ủy nhiệm để đợi AutoResetEvent đi vào
    // trạng thái signaled. Thiết lập giá trị time-out là 3 giây.
    RegisteredWaitHandle handle = ThreadPool.RegisterWaitForSingleObject(autoEvent,
    handler, state, 3000, false);
```

```
        Console.WriteLine("Press ENTER to signal the AutoResetEvent or enter \"Cancel\" to
        unregister the wait operation.");
        while (Console.ReadLine().ToUpper() != "CANCEL")
        {
            // Nếu "Cancel" không được nhập vào Console, AutoResetEvent sẽ đi vào
            // trạng thái signal, và phương thức EventHandler được thực thi.
            // AutoResetEvent sẽ tự động trở về trạng thái unsignaled.
            autoEvent.Set();
        }
        // Hủy bỏ việc đăng ký quá trình đợi.
        Console.WriteLine("Unregistering wait operation.");
        handle.Unregister(null);
        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
        }
}
```

VI.6. Phương pháp Mutex

Mutex cung cấp một cơ chế để đồng bộ hóa quá trình thực thi của các tiểu trình vượt qua biên tiến trình. Một Mutex là signaled khi nó không thuộc sở hữu của bất kỳ tiểu trình nào. Một tiểu trình giành quyền sở hữu Mutex lúc khởi dựng hoặc sử dụng một trong các phương thức được liệt kê ở trên.

Quyền sở hữu Mutex được giải phóng bằng cách gọi phương thức Mutex.ReleaseMutex (ra hiệu Mutex và cho phép một tiểu trình khác thu lấy quyền sở hữu này).

Ví dụ dưới đây sử dụng một Mutex có tên là MutexExample để bảo đảm chỉ một thể hiện của ví dụ có thể thực thi.

Chương trình MutexExample

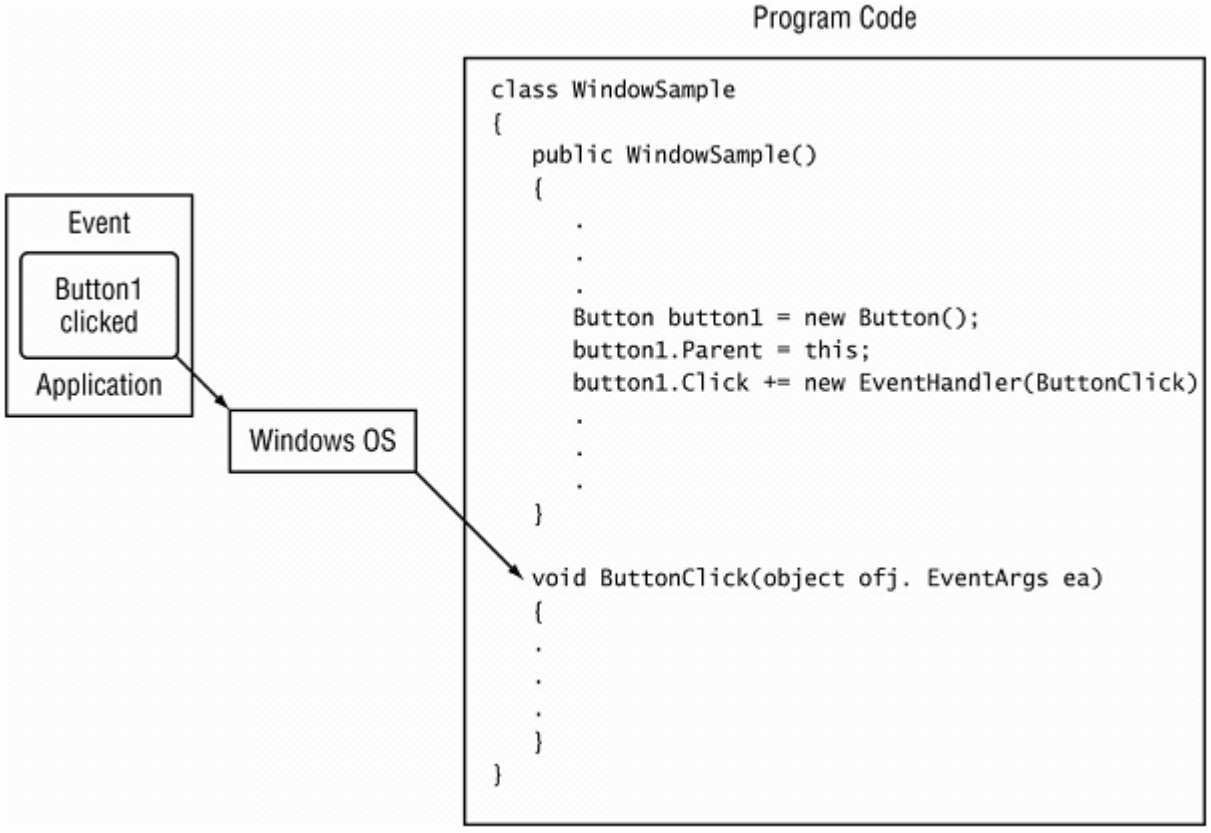
```
using System;
using System.Threading;
public class MutexExample
{
    public static void Main()
    {
        // Giá trị luận lý cho biết ứng dụng này có quyền sở hữu Mutex hay không.
        bool ownsMutex;
        // Tạo và lấy quyền sở hữu một Mutex có tên là MutexExample.
```

```
            using (Mutex mutex = new Mutex(true, "MutexExample", out ownsMutex))
            {
                // Nếu ứng dụng sở hữu Mutex, nó có thể tiếp tục thực thi;
                // nếu không, ứng dụng sẽ thoát.
                if (ownsMutex)
                {
                    Console.WriteLine("This application currently owns the mutex
                    named MutexExample. Additional instances of this application
                    will not run until you release the mutex by pressing Enter.");
                    Console.ReadLine();
                    // Giải phóng
                    Mutex.mutex.ReleaseMutex();
                }
                else
                {
                    Console.WriteLine("Another instance of this" + " application
                    already owns the mutex named" + " MutexExample. This
                    instance of the" + " application will terminate.");
                }
            }
            // Nhấn Enter để kết thúc.
            Console.WriteLine("Main method complete. Press Enter.");
            Console.ReadLine();
        }
}
```

CHƯƠNG 7. LẬP TRÌNH SOCKET BẤT ĐỒNG BỘ

VII.1. Lập trình sự kiện trong Windows

Trong lập trình sự kiện trong Windows, mỗi khi một sự kiện xảy ra, một phương thức được gọi để thực thi dựa trên sự kiện đó như trong hình dưới đây:



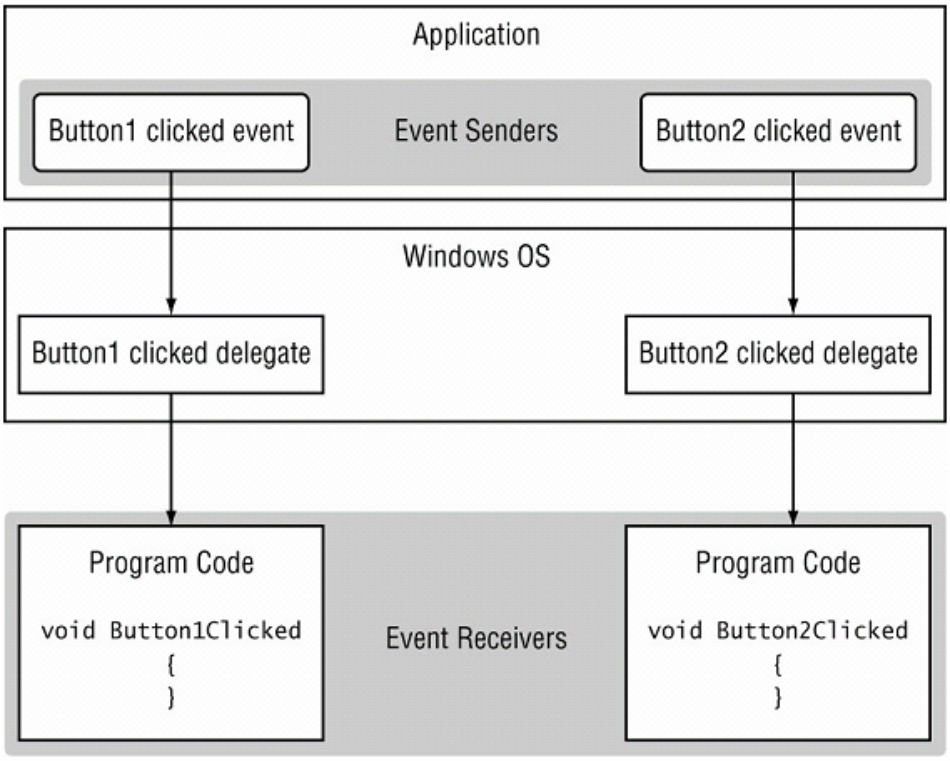
Hình VII.1: Lập trình sự kiện trên Windows

Trong các chương trước, chúng ta đã lập trình Socket trong chế độ blocking. Socket ở chế độ blocking sẽ chờ mãi mãi cho đến khi hoàn thành nhiệm vụ của nó. Trong khi nó bị blocking thì các chức năng khác của chương trình không thực hiện được.

Khi lập trình Windows thì lúc gọi một phương thức bị blocking thì toàn bộ chương trình sẽ đứng lại và không thực hiện các chức năng khác được. Do đó việc lập trình bất đồng bộ là cần thiết để cho chương trình khỏi bị đứng.

VII.1.1. Sử dụng Event và Delegate

Event là một thông điệp được gởi bởi một đối tượng mô tả một hoạt động mà nó diễn ra. Thông điệp này xác định hoạt động và truyền các dữ liệu cần thiết cho hoạt động. Event có thể là mô tả hoạt động nào đó, chẳng hạn như hoạt động click một Button, hoạt động nhận và gởi dữ liệu trên Socket. Event sender không cần thiết phải biết đối tượng nào sẽ điều khiển thông điệp sự kiện mỗi khi nó được gởi thông qua hệ thống Windows. Nó để cho bộ nhận sự kiện đăng ký với hệ thống Windows và thông báo kiểu sự kiện mà bộ nhận sự kiện muốn nhận như hình minh họa sau:



Hình VII.2: Gởi và nhận sự kiện trong Windows

Bộ nhận sự kiện được xác định trong hệ thống Windows bởi một con trỏ lớp được gọi là delegate. Một delegate là một lớp nó giữ tham chiếu đến một phương thức mà phương thức này điều khiển sự kiện được nhận. Khi hệ thống Windows nhận sự kiện. nó kiểm tra coi thử có delegate nào đăng ký để xử lý nó không. Nếu có delegate đăng ký để xử lý sự kiện, thông điệp sự kiện được truyền vào phương thức được định nghĩa bởi delegate. Sau khi phương thức hoàn tất, hệ thống Windows sẽ xử lý sự kiện tiếp theo xảy ra cho tới khi sự kiện kết thúc chương trình được phát ra.

Ví dụ đơn giản sau mô tả cách lập trình sự kiện trên Windows Forrm:

Chương trình WindowSample

```
using System;
using System.Drawing;
using System.Windows.Forms;
class WindowSample : Form
{
        private TextBox data;
        private ListBox results;
        public WindowSample()
        {
                Text = "Sample Window Program";
                Size = new Size(400, 380);
                Label label1 = new Label();
```

```
            label1.Parent = this;
            label1.Text = "Enter text string:";
            label1.AutoSize = true;
            label1.Location = new Point(10, 10);
            data = new TextBox();
            data.Parent = this;
            data.Size = new Size(200, 2 * Font.Height);
            data.Location = new Point(10, 35);
            results = new ListBox();
            results.Parent = this;
            results.Location = new Point(10, 65);
            results.Size = new Size(350, 20 * Font.Height);
            Button checkit = new Button();
            checkit.Parent = this;
            checkit.Text = "test";
            checkit.Location = new Point(235, 32);
            checkit.Size = new Size(7 * Font.Height, 2 * Font.Height);
            checkit.Click += new EventHandler(checkit_OnClick);
        }
        void checkit_OnClick(object obj, EventArgs ea)
        {
            results.Items.Add(data.Text);
            data.Clear();
        }
        public static void Main()
        {
            Application.Run(new WindowSample());
        }
    }
```

Điểm chính trong chương trình này là EvenHandler đăng ký phương thức ButtonOnClick() cho đối tượng Button với sự kiện click:

checkit.Click += new EventHandler(checkit_OnClick);

Khi người dùng click button, điều khiển chương trình sẽ được chyển đến phương thức ButtonOnClick()

void checkit_OnClick(object obj, EventArgs ea)

```
{
        results.Items.Add(data.Text);
        data.Clear();
}
```

VII.1.2. Lớp AsyncCallback trong lập trình Windows

Khi sự kiện kích hoạt delegate, .NET cung cấp một cơ chế để kích hoạt delegate. Lớp AsyncCallback cung cấp các phương thức để bắt đầu một chức năng bất đồng bộ và cung cấp một phương thức delegate để gọi khi chức năng bất đồng bộ kết thúc.

Tiến trình này khác với cách lập trình sự kiện cơ bản, sự kiện này không phát sinh ra từ đối tượng Windows mà nó xuất phát từ một phương thức khác trong chương trình. Phương thức này chính nó đăng ký một delegate AsyncCallback để gọi khi phương thức hoàn tất chức năng của nó.

Socket sử dụng phương thức được định nghĩa trong lớp AsyncCallback để cho phép các chức năng mạng hoạt động một cách bất đồng bộ. Nó phát tín hiệu cho hệ điều hành khi chức năng mạng hoàn tất. Trong môi trường lập trình Windows, những phương thức này giúp cho ứng dụng khỏi bị treo trong khi chờ các chức năng mạng hoàn tất.

VII.2. Sử dụng Socket bất đồng bộ

Đối tượng Socket có nhiều phương thức sử dụng lớp AsyncCallback để gọi các phương thức khác khi các chức năng mạng hoàn tất. Nó cho phép dùng tiếp tục xử lý các sự kiện khác trong khi chờ cho các chức năng mạng hòa thành công việc của nó.

Các phương thức bất đồng bộ của Socket chia các chức năng mạng làm hai phần:

- Một phương thức Begin bắt đầu các chức năng mạng và đăng ký phương thức AsyncCallback.
- Một phương thức End hoàn thành chức năng mạng khi phương thức AsyncCallback được gọi.

Bảng sau cho biết các phương thức bất đồng bộ có thể được giữa với Socket. Mỗi phương thức Begin được kết hợp với một phương thức End để hoàn tất chức năng của chúng:

Phương thức bắt đầu	**Mô tả**	**Phương thức kết thúc**
BeginAccept()	Chấp nhận kết nối	EndAccept()
BeginConnect()	Kết nối đến thiết bị ở xa	EndConnect()
BeginReceive()	Nhận dữ liệu từ Socket	EndReceive()
BeginReceiveFrom()	Nhận dữ liệu từ thiết bị ở xa	EndReceiveFrom()
BeginSend()	Gởi dữ liệu từ Socket	EndSend()
BeginSendTo()	Gởi dữ liệu đến thiết bị ở xa	EndSendTo()

VII.2.1. Thành lập kết nối

Phương thức được dùng để thành lập kết nối với thiết bị ở xa phụ thuộc vào chương trình đóng vai trò là Server hay Client. Nếu là Server thì phương thức BeginAccept() được dùng, nếu là Client thì phương thức BeginConnect() được dùng.

VII.2.1.1. Phương thức BeginAccept() và EndAccept()

Để chấp nhận kết nối từ thiết bị ở xa, phương thức BeginAccetp() được dùng. Định dạng của nó như sau:

IAsyncResult BeginAccept(AsyncCallback callback, object state)

Phương thức BeginAccept() nhận hai đối số: tên của phương thức AsyncCallback dùng để kết thúc phương thức và một đối tượng state có thể được dùng để truyền thông tin giữa các phương thức bất đồng bộ. Phương thức này được dùng như sau:

```
Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);

sock.Bind(iep);

sock.Listen(5);

sock.BeginAccept(new AsyncCallback(CallAccept), sock);
```

Đoạn code ở trên tạo ra một đối tượng Socket và gán nó đến một địa chỉ IP cục bộ và một port TCP cục bộ để chấp nhận kết nối. Phương thức BeginAccept() định nghĩa delegate được dùng khi có kết nối trên Socket. Tham số cuối cùng được truyền vào phương thức BeginAccept() là Socket ban đầu được tạo ra.

Sau khi phương thức BeginAccept() kết thúc, phương thức AsyncCallback định nghĩa sẽ được gọi khi kết nối xảy ra. Phương thức AsyncCallback phải bao gồm phương thức EndAccept() để kết thúc việc chấp nhận Socket. Sau đây là định dạng của phương thức EndAccept():

```
Socket EndAccept(IAsyncResult iar);
```

Đối tượng IasyncResult truyền giá trị bất đồng bộ từ phương thức BeginAccept() đến phương thức EndAccept(). Cũng giống như phương thức đồng bộ Accept(), phương thức EndAccept() trả về một đối tượng Socket được dùng để kết nối với Client. Tất cả các thông tin liên lạc vói thiết bị ở xa đều được thực hiện thông qua đối tượng Socket này.

```
private static void CallAccept(IAsyncResult iar)

{

Socket server = (Socket)iar.AsyncState;

Socket client = server.EndAccept(iar);

.

.

.

}
```

Tên của phương thức AsyncCallback phải giống với tên của phương thức được dùng làm tham số của phương thức BeginAccept(). Bước đầu tiên trong phương thức là nhận Socket ban đầu của Server. Việc này được thực hiện bằng cách dùng property AsyncState của lớp IasyncResult. Property này có kiểu là object do đó nó phải được ép kiểu sang một đối tượng Socket.

Sau khi Socket ban đầu được lấy về, phương thức EndAccept() có thể lấy được đối tượng Socket mới để truyền thông với thiết bị ở xa. Đối số truyền vào của của phương thức EndAccept() phải giống với đối số truyền vào phương thức AsyncCallback.

Đối tượng Client Socket sau khi được tạo ra, nó có thể được dùng giống như bất kỳ đối tượng Socket nào khác, nó có thể được sử dụng các phương thức đồng bộ hoặc bất đồng bộ để gởi và nhận dữ liệu.

VII.2.1.2. Phương thức BeginConnect() và EndConnect()

Để ứng dụng Client kết nối đến Server ở xa bằng phương thức bất đồng bộ ta phải dùng phương thức BeginConnect(). Định dạng của phương thức này như sau:

IAsyncResult BeginConnect(EndPoint ep, AsyncCallback callback, Object state)

Tham số truyền vào phương thức BeginConnect() là một EndPoint của thiết bị ở xa cần kết nối đến. Giống như phương thức BeginAccept(), phương thức BeginConnect() cũng chỉ ra tên phương thức do delegate AsyncCallback tham chiếu đến và phương thức này được gọi khi kết nối hoàn tất. Tham số cuối cùng là một đối tượng state, nó có thể được truyền vào phương thức EndConnect() để truyền tải các dữ liệu cần thiết.

Đây là đoạn code ví dụ của phương thức BeginConnect():

```
Socket newsock = new Socket(AddressFamily.InterNetwork,
SocketType.Stream,
ProtocolType.Tcp);
IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
newsock.BeginConnect(iep, new AsyncCallback(Connected), newsock);
```

Đoạn code này tạo ra một đối tượng Socket newsock và một đối tượng IPEndPoint iep cho thiết bị ở xa. Phương thức BeginConnect() tham chiếu đến phương thức AsyncCallback (Connect) và truyền đối tượng Socket ban đầu newsock đến phương thức AsyncCallback.

Khi kết nối được thành lập phương thức do delegate AsyncCallback tham chiếu tới được gọi. Phương thức này dùng phương thức EndConnect() để hoàn thành việc kết nối. Định dạng của phương thức EndConnect() như sau:

EndConnect(IAsyncResult iar)

Đối tượng IasyncResult truyền giá trị từ phương thức BeginConnect(). Sau đây là ví dụ cách dùng phương thức này:

```
public static void Connected(IAsyncResult iar)
{
        Socket sock = (Socket)iar.AsyncState;
        try
        {
                sock.EndConnect(iar);
        } catch (SocketException)
        {
                Console.WriteLine("Unable to connect to host");
        }
}
```

Đầu tiên ta lấy Socket ban đầu được sử dụng bởi phương thức BeginConnect(), Socket này lấy được là nhờ vào thuộc tính object của đối tượng IAsyncResult được truyền vào trong phương thức tham chiếu bởi delegate AsyncCallback.

Sau khi Socket ban đầu được tạo ra, phương thức EndConnect() được gọi, đối số truyền vào phương thức EndConnect() là một đối tượng IAsyncResult chỏ ngược trở lại phương thức BeginConnect() ban đầu. Bởi vì thiết bị ở xa có thể kết nối được và cũng có thể không nên tốt nhất là đặt nó vào trong khối try – catch.

VII.2.2. Gởi dữ liệu

VII.2.2.1. Phương thức BeginSend() và phương thức EndSend()

Phương thức BeginSend() cho phép gởi dữ liệu đến một Socket đã được kết nối.

Định dạng của phương thức này như sau:

```
IAsyncResult BeginSend(byte[] buffer, int offset, int size, SocketFlags sockflag,
AsyncCallback callback, object state)
```

Hầu hết các đối số của phương thức này giống như các đối số của phương thức đồng bộ Send() chỉ có thêm hai đối số là AsyncCallback và object.

- Đối số AsyncCallback: xác định phương thức được gọi khi phương thức BeginSend() thực hiện thành công.
- Đối số object: gởi thông tin tình trạng đến phương thức EndSend()

Sau đây là một ví dụ của phương thức BeginSend():

```
sock.BeginSend(data, 0, data.Length, SocketFlags.None, new AsyncCallback(SendData),
sock);
```

Trong ví dụ này toàn bộ dữ liệu trong bộ đệm data được gởi đi và phương thức SendData được gọi khi Socket sẵng sàng gởi dữ liệu. Đối tượng Socket sock sẽ được truyền đến phương thức do delegate AsyncCallback tham chiếu tới.

Phương thức EndSend() sẽ hoàn tất việc gởi dữ liệu. Định dạng của phương thức này như sau:

```
int EndSend(IAsyncResult iar)
```

Phương thức EndSend() trả về số byte được gởi thành công thừ Socket. Sau đây là một ví dụ của phương thức EndSend():

```
private static void SendData(IAsyncResult iar)
{
    Socket server = (Socket)iar.AsyncState;
    int sent = server.EndSend(iar);
}
```

Socket ban đầu được tạo lại bằng cách dùng thuộc tính AsyncState của đối tượng IAsyncResult.

VII.2.2.2. Phương thức BeginSendTo() và EndSendTo()

Phương thức BeginSendTo() được dùng với Socket phi kết nối để bắt đầu truyền tải dữ liệu bất đồng bộ tói thiết bị ở xa. Định dạng của phương thức BeginSendTo() như sau:

IAsyncResult BeginSendTo(byte[] buffer, int offset, int size, SocketFlags sockflag, EndPoint ep, AsyncCallback callback, object state)

Các đối số của phương thức BeginSendTo() cũng giống như các đối số của phương thức SendTo().

Sau đây là một ví dụ của phương thức BeginSendTo():

IPEndPoint iep = new IPEndPoint(IPAddress.Parse("192.168.1.6"), 9050);

sock.BeginSendTo(data, 0, data.Length, SocketFlags.None, iep, new AsynCallback(SendDataTo), sock);

Phương thức SendDataTo() do delegate AsyncCallback tham chiếu đến được truyền vào làm đối số của phương thức BeginSendTo(). Phương thức này sẽ được thực thi khi dữ liệu bắt đầu được gởi đi từ Socket. Phương thức EndSendTo() sẽ được thực thi khi quá trình gởi dữ liệu kết thúc. Định dạng của phương thức này như sau:

int EndSendTo(IAsyncResult iar);

Đối số truyền vào của phương thức EndSendTo() là một đối tượng IAsyncResult, đối tượng này mang giá trị được truyền từ phương thức BeginSendTo().

Khi quá trình gởi dữ liệu bất đồng bộ kết thúc thì phương thức EndSendTo() sẽ trả về số byte mà nó thực sự gởi được.

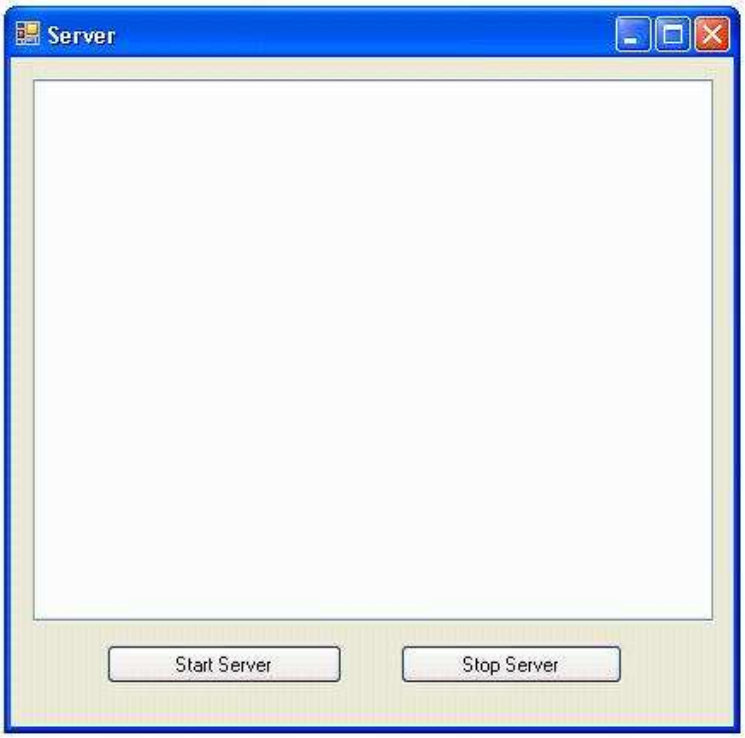
VII.2.3. Nhận dữ liệu

Bao gồm phương thức BeginReceive(), EndReceive, BeginReceiveFrom(), EndReceiveFrom()

Cách sử dụng của các phương thức này cũng tương tự như cách sử dụng của các phương thức: BeginSend(), EndSend(), BeginSendTo() và EndSendTo()

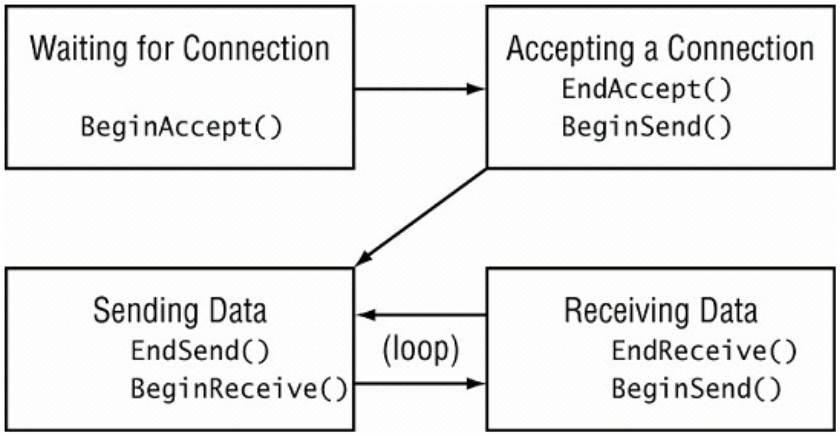
VII.2.4. Chương trình WinForm gởi và nhận dữ liệu giữa Client và Server

VII.2.4.1. Chương trình Server



Hình VII.3: Giao diện Server

VII.2.4.2. Mô hình chương trình Server



Hình VI.4: Mô hình chương trình Server

VII.2.4.3. Lớp ServerProgram

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
```

```
using System.Net.Sockets;
namespace Server
{
class ServerProgram
{
        private IPAddress serverIP;
        public IPAddress ServerIP
        {
                get
                {
                        return serverIP;
                }
                set
                {
                        this.serverIP = value;
                }
        }
        private int port;
        public int Port
        {
                get
                {
                        return this.port;
                }
                set
                {
                        this.port = value;
                }
        }
        //delegate để set dữ liệu cho các Control
        //Tại thời điểm này ta chưa biết dữ liệu sẽ được hiển thị vào đâu nên ta phải dùng
        delegate
        public delegate void SetDataControl(string Data);
        public SetDataControl SetDataFunction = null;
        Socket serverSocket = null ;
```

```
IPEndPoint serverEP = null;
Socket clientSocket = null;
//buffer để nhận và gởi dữ liệu
byte[] buff = new byte[1024];
//Số byte thực sự nhận được
int byteReceive = 0;
string stringReceive = "";
public ServerProgram(IPAddress ServerIP, int Port)
{
        this.ServerIP = ServerIP;
        this.Port = Port;
}
//Lắng nghe kết nối
public void Listen()
{
        serverSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
        ProtocolType.Tcp);
        serverEP = new IPEndPoint(ServerIP, Port);
        //Kết hợp Server Socket với Local Endpoint
        serverSocket.Bind(serverEP);
        //Lắng nghe kết nối trên Server Socket
        //-1: không giới hạn số lượng client kết nối đến
        serverSocket.Listen(-1);
        SetDataFunction("Dang cho ket noi");
        //Bắt đầu chấp nhận Client kết nối đến
        serverSocket.BeginAccept(new AsyncCallback(AcceptScoket), serverSocket);
}
//Hàm callback chấp nhận Client kết nối
private void AcceptScoket(IAsyncResult ia)
{
        Socket s = (Socket)ia.AsyncState;
        //Hàm Accept sẽ block server lại và chờ Client kết nối đến
        //Sau khi Client kết nối đến sẽ trả về socket chứa thông tin của Client
        clientSocket = s.EndAccept(ia);
        string hello = "Hello Client";
```

```
            buff = Encoding.ASCII.GetBytes(hello);
            SetDataFunction("Client " + clientSocket.RemoteEndPoint.ToString() + "da
            ket noi den");
            clientSocket.BeginSend(buff, 0, buff.Length, SocketFlags.None, new
            AsyncCallback(SendData), clientSocket);
        }
        private void SendData(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            s.EndSend(ia);
            //khởi tạo lại buffer để nhận dữ liệu
            buff = new byte[1024];
            //Bắt đầu nhận dữ liệu
            s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
            AsyncCallback(ReceiveData), s);
        }
        public void Close()
        {
            clientSocket.Close();
            serverSocket.Close();
        }
        private void ReceiveData(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            try
            {
                //Hàm EmdReceive sẽ bị block cho đến khi có dữ liệu trong TCP
                buffer byteReceive = s.EndReceive(ia);
            }
            catch
            {
                //Trường hợp lỗi xảy ra khi Client ngắt kết nối
                this.Close();
                SetDataFunction("Client ngat ket noi");
                this.Listen();
                return;
```

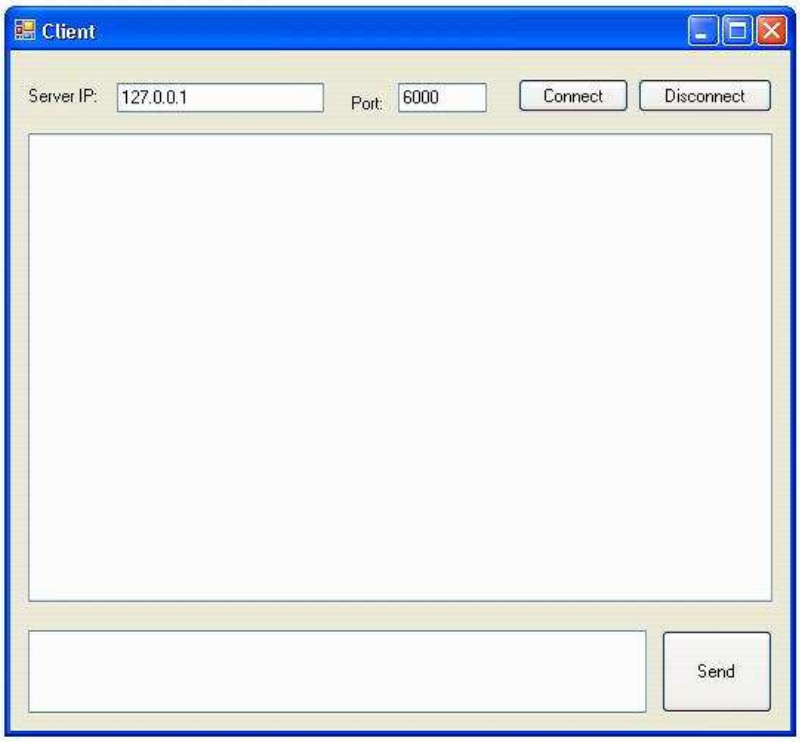
```
                }
                //Nếu Client shutdown thì hàm EndReceive sẽ trả về 0
                if (byteReceive == 0)
                {
                        Close();
                        SetDataFunction("Clien dong ket noi");
                }
                else
                {
                        stringReceive = Encoding.ASCII.GetString(buff, 0, byteReceive);
                        SetDataFunction(stringReceive);
                        //Sau khi Server nhận dữ liệu xong thì bắt đầu gởi dữ liệu xuống cho
                        Client
                        s.BeginSend(buff, 0, buff.Length, SocketFlags.None, new
                        AsyncCallback(SendData), s);
                }
        }
}
}
```

VII.2.4.4. Lớp ServerForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
namespace Server
{
        public partial class ServerForm : Form
        {
                ServerProgram server = new ServerProgram(IPAddress.Any, 6000);
                public ServerForm()
```

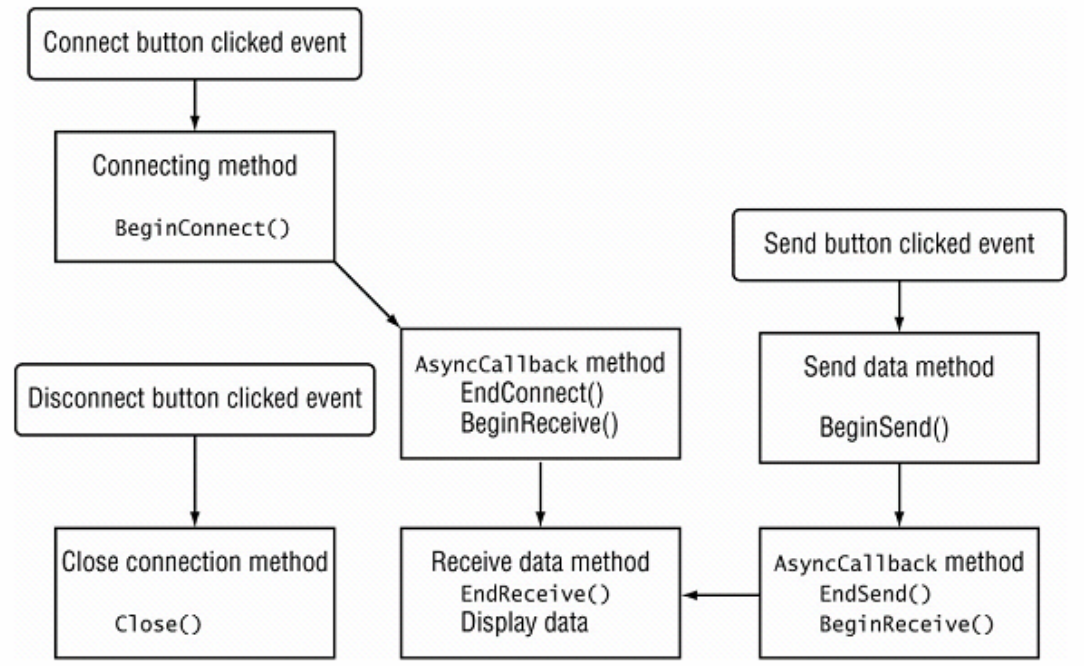
```
        {
            InitializeComponent();
            CheckForIllegalCrossThreadCalls = false;
            server.SetDataFunction = new
            ServerProgram.SetDataControl(SetData);
        }
        private void SetData(string Data)
        {
            this.listBox1.Items.Add(Data);
        }
        private void cmdStart_Click(object sender, EventArgs e)
        {
            server.Listen();
        }
        private void cmdStop_Click(object sender, EventArgs e)
        {
            this.server.Close();
            SetData("Server dong ket noi");
        }
        private void ServerForm_Load(object sender, EventArgs e)
        {
        }
    }
}
```

VII.2.5. Chương trình Client



Hình VII.5: Giao diện Client

VII.2.5.1. Mô hình chương trình Client



Hình VII.6: Mô hình chương trình Client

VII.2.5.2. Lớp ClientProgram

```
using System;
```

```
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
namespace Client
{
class ClientProgram
{
        //delegate để set dữ liệu cho các Control
        //Tại thời điểm này ta chưa biết dữ liệu sẽ được hiển thị vào đâu nên ta phải dùng delegate
        public delegate void SetDataControl(string Data);
        public SetDataControl SetDataFunction = null;
        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //Số byte thực sự nhận được
        int byteReceive = 0;
        //Chuỗi nhận được
        string stringReceive = "";
        Socket serverSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        IPEndPoint serverEP = null;
        //Lắng nghe kết nối
        public void Connect(IPAddress serverIP, int Port)
        {
                serverEP = new IPEndPoint(serverIP, Port);
                //Việc kết nối có thể mất nhiều thời gian nên phải thực hiện bất đồng bộ
                serverSocket.BeginConnect( serverEP, new AsyncCallback(ConnectCallback), serverSocket);
        }
        //Hàm callback chấp nhận Client kết nối
        private void ConnectCallback(IAsyncResult ia)
        {
                //Lấy Socket đang thực hiện việc kết nối bất đồng bộ
                Socket s = (Socket)ia.AsyncState;
                try
```

```
        {
            //Set dữ liệu cho Control
            SetDataFunction("Đang chờ kết nối");
            //Hàm EndConnect sẽ bị block cho đến khi kết nối thành công
            s.EndConnect(ia);
            SetDataFunction("Kết nối thành công");
        }
        catch
        {
            SetDataFunction("Kết nối thất bại");
            return;
        }
        //Ngay sau khi kết nối xong bắt đầu nhận câu chào từ Server gởi xuống
        s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
        AsyncCallback(ReceiveData), s);
    }
    private void ReceiveData(IAsyncResult ia)
    {
        Socket s = (Socket)ia.AsyncState;
        byteReceive = s.EndReceive(ia);
        stringReceive = Encoding.ASCII.GetString(buff, 0, byteReceive);
        SetDataFunction(stringReceive);
    }
    private void SendData(IAsyncResult ia)
    {
        Socket s = (Socket)ia.AsyncState;
        s.EndSend(ia);
        //khởi tạo lại buffer để nhận dữ liệu
        buff = new byte[1024];
        //Bắt đầu nhận dữ liệu
        s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
        AsyncCallback(ReceiveData), s);
    }
    //Hàm ngắt kết nối
    public bool Disconnect()
```

```
        {
            try
            {
                //Shutdown Soket đang kết nối đến Server
                serverSocket.Shutdown(SocketShutdown.Both);
                serverSocket.Close();
                return true;
            }
            catch
            {
                return false;
            }
        }
        //Hàm gởi dữ liệu
        public void SendData(string Data)
        {
            buff = Encoding.ASCII.GetBytes(Data);
            serverSocket.BeginSend(buff, 0, buff.Length, SocketFlags.None, new
            AsyncCallback(SendToServer), serverSocket);
        }
        //Hàm CallBack gởi dữ liệu
        private void SendToServer(IAsyncResult ia)
        {
            Socket s = (Socket)ia.AsyncState;
            s.EndSend(ia);
            buff = new byte[1024];
            s.BeginReceive(buff, 0, buff.Length, SocketFlags.None, new
            AsyncCallback(ReceiveData), s);
        }
    }
    }
```

VII.2.5.3. Lớp ClientForm

```
    using System;
    using System.Collections.Generic;
    using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
namespace Client
{
	public partial class ClientForm : Form
	{
		ClientProgram client = new ClientProgram();
		public ClientForm()
		{
			InitializeComponent();
			CheckForIllegalCrossThreadCalls = false;
			client.SetDataFunction = new ClientProgram.SetDataControl(SetData);
		}
		private void SetData(string Data)
		{
			this.listBox1.Items.Add(Data);
		}
		private void cmdConnect_Click(object sender, EventArgs e)
		{
			client.Connect(IPAddress.Parse(this.txtServerIP.Text),
			int.Parse(this.txtPort.Text));
		}
		private void cmdDisconnect_Click(object sender, EventArgs e)
		{
			client.Disconnect();
		}
		private void cmdSend_Click_1(object sender, EventArgs e)
		{
			client.SendData(this.txtInput.Text);
			this.txtInput.Text = "";
		}
```

```
        }
    }
```

VII.3. Lập trình Socket bất đồng bộ sử dụng tiểu trình

VII.3.1. Lập trình sử dụng hàng đợi gởi và hàng đợi nhận thông điệp

Trong cách lập trình này, chúng ta sẽ dùng hai hàng đợi, một hàng đợi gởi và một hàng đợi nhận để thực hiện việc gởi và nhận dữ liệu. Lớp Queue nằm trong namespace System.Collections giúp ta thực hiện việc này.

```
Queue inQueue = new Queue();
Queue outQueue = new Queue();
```

Hai phương thức quan trọng của lớp Queue được dùng trong lập trình mạng là EnQueue() và DeQueue(). Phương thức EnQueue() sẽ đưa một đối tượng vào hàng đợi và phương thức DeQueue() sẽ lấy đối tượng đầu tiên từ hàng đợi ra.

Ý tưởng của phương pháp lập trình này là ta sẽ dùng hai vòng lặp vô hạn để kiểm tra hàng đợi gởi và hàng đợi nhận, khi hàng đợi gởi có dữ liệu thì dữ liệu đó được gởi ra ngoài mạng thông qua Socket, tương tự khi hàng đợi nhận có dữ liệu thì nó sẽ ngay lập tức lấy dữ liệu đó ra và xử lý.

```
private void Send()
{
        while (true)
        {
                if (OutQ.Count > 0)
                {
                        streamWriter.WriteLine(OutQ.Dequeue().ToString());
                        streamWriter.Flush();
                }
        }
}
private void Receive()
{
        string s;
        while (true)
        {
                s = streamReader.ReadLine();
                InQ.Enqueue(s);
        }
}
```

Để chạy song song hai vòng lặp vô hạn này ta phải tạo ra hai tiểu trình riêng, mỗi vòng lặp vô hạn sẽ chạy trong một tiểu trình riêng biệt do đó trong khi hai vòng lặp vô hạn này chạy thì tiến trình chính vẫn có thể làm được các công việc khác.

```
Thread tSend = new Thread(new ThreadStart(Send));
tSend.Start();
Thread tReceive = new Thread(new ThreadStart(Receive));
tReceive.Start();
```

Ngoài ra, ta còn sử dụng một tiểu trình khác thực hiện việc kết nối nhằm tránh gây treo tiến trình chính.

```
Thread tConnect = new Thread(new ThreadStart(WaitingConnect));
tConnect.Start();
```

Việc điều khiển kết nối được thực hiện trong một tiểu trình khác và được xử lý bằng phương thức WaitingConnect():

```
private void WaitingConnect()
{
        tcpListener = new TcpListener(1001);
        tcpListener.Start();
        socketForClient = tcpListener.AcceptSocket();
        if (socketForClient.Connected)
        {
                MessageBox.Show("Client Connected");
                netWorkStream = new NetworkStream(socketForClient);
                streamReader = new StreamReader(netWorkStream);
                streamWriter = new StreamWriter(netWorkStream);
                tSend = new Thread(new ThreadStart(Send));
                tSend.Start();
                tReceive = new Thread(new ThreadStart(Receive));
                tReceive.Start();
        }
        else
        {
                MessageBox.Show("Ket noi khong thanh cong");
        }
}
```

Việc nhập dữ liệu được thực hiện bằng phương thức InputData()

```
public void InputData(string s)
```

```
{
        InQ.Enqueue(s);
        OutQ.Enqueue(s);
}
```

Phương thức này đơn giản chỉ đưa dữ liệu nhập vào hàng đợi nhận để hiển thị dữ liệu lên màn hình và đưa dữ liệu nhập này vào hàng đợi gởi để gởi ra ngoài mạng.

Lớp ServerObject

```
using System;
using System.IO;
using System.Windows.Forms;
using System.Threading;
using System.Collections;
using System.Net.Sockets;
using System.Collections.Generic;
using System.Text;
namespace Server
{
        class ServerObject
        {
                Thread tSend, tReceive, tConnect;
                public Queue InQ = new Queue();
                public Queue OutQ = new Queue();
                private TcpListener tcpListener;
                Socket socketForClient;
                private NetworkStream netWorkStream;
                private StreamWriter streamWriter;
                private StreamReader streamReader;
                public void CreateConnection()
                {
                        tConnect = new Thread(new ThreadStart(WaitingConnect));
                        tConnect.Start();
                }
                private void WaitingConnect()
                {
                        tcpListener = new TcpListener(1001);
```

```
            tcpListener.Start();
            socketForClient = tcpListener.AcceptSocket();
            if (socketForClient.Connected)
            {
                MessageBox.Show("Client Connected");
                netWorkStream = new NetworkStream(socketForClient);
                streamReader = new StreamReader(netWorkStream);
                streamWriter = new StreamWriter(netWorkStream);
                tSend = new Thread(new ThreadStart(Send));
                tSend.Start();
                tReceive = new Thread(new ThreadStart(Receive));
                tReceive.Start();
            }
            else
            {
                MessageBox.Show("Ket noi khong thanh cong");
            }
            //socketForClient.Close();
        }
        private void Send()
        {
            while (true)
            {
                if (OutQ.Count > 0)
                {
                streamWriter.WriteLine(OutQ.Dequeue().ToString());
                streamWriter.Flush();
                }
            }
        }
        private void Receive()
        {
            string s;
            while (true)
            {
```

```
                    s = streamReader.ReadLine();
                    InQ.Enqueue(s);
                }
            }
            public void InputData(string s)
            {
                InQ.Enqueue(s);
                OutQ.Enqueue(s);
            }
        }
    }
```

Lớp ClientObject

```
    using System;
    using System.Windows.Forms;
    using System.Collections;
    using System.Net.Sockets;
    using System.Threading;
    using System.IO;
    using System.Collections.Generic;
    using System.Text;
    namespace Client
    {
        class ClientObject
        {
        Thread tSend, tReceive, tConnect;
        public Queue InQ = new Queue();
        public Queue OutQ = new Queue();
        private TcpClient socketForServer;
        private NetworkStream networkStream;
        private StreamWriter streamWriter;
        private StreamReader streamReader;
        public void Connect()
        {
            tConnect = new Thread(new ThreadStart(WaitConnect));
            tConnect.Start();
```

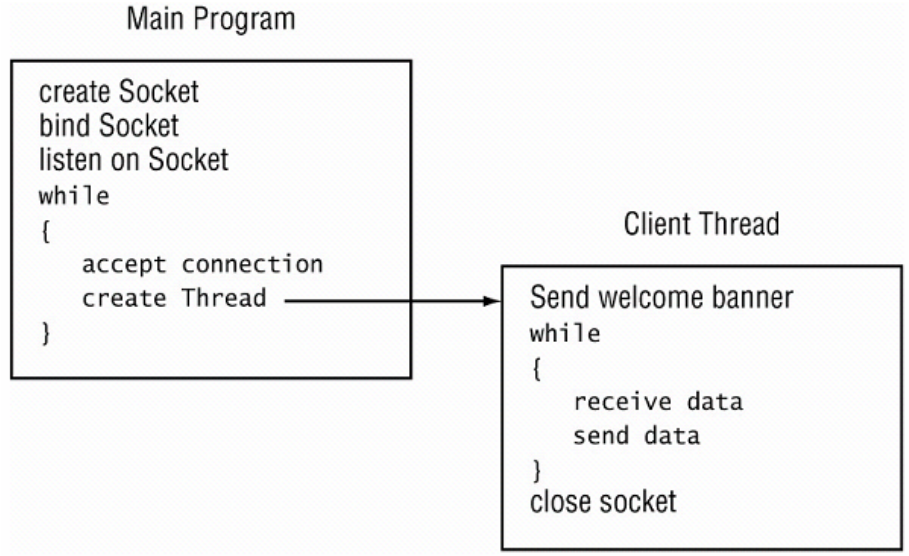
```
}
public void WaitConnect()
{
        try
        {
                socketForServer = new TcpClient("localhost", 1001);
        }
        catch {
                MessageBox.Show("Ket noi that bai");
        }
        networkStream = socketForServer.GetStream();
        streamReader = new StreamReader(networkStream);
        streamWriter = new StreamWriter(networkStream);
        tSend = new Thread(new ThreadStart(Send));
        tSend.Start();
        tReceive = new Thread(new ThreadStart(Receive));
        tReceive.Start();
}
private void Send()
{
        while (true)
        {
                if (OutQ.Count > 0)
                {
                        streamWriter.WriteLine(OutQ.Dequeue().ToString());
                        streamWriter.Flush();
                }
        }
}
private void Receive()
{
        string s;
        while (true)
        {
                s = streamReader.ReadLine();
```

```
                InQ.Enqueue(s);
            }
        }
        public void InputData(string s)
        {
            InQ.Enqueue(s);
            OutQ.Enqueue(s);
        }
        }
}
```

VII.3.2. Lập trình ứng dụng nhiều Client

Một trong những khó khăn lớn nhất của các nhà lập trình mạng đó là khả năng xử lý nhiều Client kết nối đến cùng một lúc. Để ứng dụng server xử lý được với nhiều Client đồng thời thì mỗi Client kết nối tới phải được xử lý trong một tiểu trình riêng biệt.

Mô hình xử lý như sau:



Hình VII.7: Mô hình xử lý một Server nhiều Client

Chương trình Server sẽ tạo ra đối tượng Socket chính trong chương trình chính, mỗi khi Client kết nối đến, chương trình chính sẽ tạo ra một tiểu trình riêng biệt để điều khiển kết nối. Bởi vì phương thức Accept() tạo ra một đối tượng Socket mới cho mỗi kết nối nên đối tượng mới này được sử dùng để thông tin liên lạc với Client trong tiểu trình mới. Socket ban đầu được tự do và có thể chấp nhận kết nối khác.

Chương trình ThreadedTcpSrvr

```
using System;
using System.Net;
using System.Net.Sockets;
```

```
using System.Text;
using System.Threading;
using System.Net;
using System.Net.Sockets;
class ThreadedTcpSrvr
{
        private TcpListener client;
        public ThreadedTcpSrvr()
        {
                client = new TcpListener(5000);
                client.Start();
                Console.WriteLine("Dang cho client...");
                while (true)
                {
                        while (!client.Pending())
                        {
                                Thread.Sleep(1000);
                        }
                        ConnectionThread newconnection = new ConnectionThread();
                        newconnection.threadListener = this.client;
                        Thread newthread = new Thread(new
                        ThreadStart(newconnection.HandleConnection));
                        newthread.Start();
                }
        }
        public static void Main()
        {
                ThreadedTcpSrvr server = new ThreadedTcpSrvr();
        }
}
class ConnectionThread
{
        public TcpListener threadListener;
        private static int connections = 0;
        public void HandleConnection()
```

```
        {
            int recv;
            byte[] data = new byte[1024];
            TcpClient client = threadListener.AcceptTcpClient();
            NetworkStream ns = client.GetStream();
            connections++;
            Console.WriteLine("Client moi duoc chap nhan, Hien tai co {0} ket noi",
            connections);
            string welcome = "Xin chao client";
            data = Encoding.ASCII.GetBytes(welcome);
            ns.Write(data, 0, data.Length);
            while (true)
            {
                data = new byte[1024];
                recv = ns.Read(data, 0, data.Length);
                if (recv == 0) break;
                ns.Write(data, 0, recv);
            }
            ns.Close(); client.Close(); connection--;
            Console.WriteLine("Client disconnected: {0} active connections",
            connections);
        }
}
```

TÀI LIỆU THAM KHẢO

[1] C Sharp Network Programming, Sybex

[2] Microsoft Network Programming For The Microsoft Dot Net Framework, Microsoft Press

[3] Network programming in dot NET C Sharp and Visual Basic dot NET, Digital Press