การศึกษาและพัฒนาโปรแกรม WWW proxy ที่สามารถส่งข้อมูลติดต่อกับ application

The Studying and Developing a WWW proxy for Interfacing with Applications

โดย

นายอภิชาติ ศรีเพียร

รหัส 41067124

H001692

อาจารย์ที่ปรึกษา

ดร. โชติพัชร์ ภรณวลัย

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน
หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
ภาคเรียนที่ 1 ปีการศึกษา 2543
คณะเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	การศึกษาและพัฒนาโปรแกรม WWW proxy ที่สามารถส่งข้อมูลติดต่อกับ application
นักศึกษา	นายอภิชาติ ศรีเพียร
อาจารย์ที่ปรึกษา	ดร. โชติพัชร์ ภรณวลัย
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
ปีการศึกษา	2543

บทคัดย่อ

โปรแกรม WWW proxy เป็นโปรแกรมที่ถูกพัฒนาขึ้นมาเพื่อตอบสนองต่อความต้องการใช้งาน world wide web ที่ได้รับความนิยมอย่างสูง โดย WWW proxy ที่ใช้งานในปัจจุบันนั้น ส่วนใหญ่ถูกพัฒนาขึ้นเพื่อช่วยให้ การใช้บริการ world wide web มีประสิทธิภาพและมีความรวดเร็วมากขึ้น แต่เพียงเพื่อเรียกใช้และแสดงผลผ่านทาง WWW browser เท่านั้น ในการศึกษานี้จะทำการศึกษาถึงวิธีและทำการพัฒนาโปรแกรม WWW proxy โดยจะมุ่งเน้นที่ความสามารถในการส่งข้อมูลเอกสาร (web page) ที่ได้รับ และติดต่อไปยัง application อื่นที่ทำงานบนระบบปฏิบัติการ windows เพื่อให้สามารถนำข้อมูลที่ได้มานั้นไปประมวลผลโดย application เหล่านั้นได้ เป็นการเพิ่มความสามารถในการปรับเปลี่ยนข้อมูลเอกสารที่จะแสดงผ่านทาง WWW browser ได้ ในลักษณะต่างๆ ตามที่ผู้ใช้ต้องการ ทั้งนี้ ขึ้นกับลักษณะการประมวลผลของ application ที่เลือกใช้นั้นๆ

Title	The Studying and Developing a WWW proxy for Interfacing with Applications
Student	Mr. Apichart Sripian
Advisor	Dr. Chotipat Pornnawalai
Level of Study	Master of Science in Information Technology
Major	Information Science
Academic Year	2000

ABSTRACT

WWW proxy program was developed because if the need of using world wide web. It is used to make the accessing the world wide web more efficiently and faster by using the caching technique. But in this way, all information which are passed through the WWW proxy are only provided for display by WWW browser. Although we can process these information to make other useful information before send them to display by WWW browser. In this paper, I will study and develop a WWW proxy program that connect to other application and send data (web page) between them. And improve the ability of WWW proxy program which can interface with other difference applications.

สารบัญ

หน้า

บทคัดย่อภาษาไทย I

บทคัดย่อภาษาอังกฤษ II

สารบัญ III

1. บทนำ 1

1.1 วัตถุประสงค์ 2

1.2 ขอบเขตของโครงการ 2

1.3 ผลที่คาดว่าจะได้รับ 2

1.4 เนื้อหาของบทความ 2

2. WORLD WIDE WEB และ HYPER TEXT TRANSFER PROTOCOL 4

2.1 WORLD WIDE WEB 4

2.2 HYPER TEXT TRANSFER PROTOCOL (HTTP) 4

2.3 HTTP MESSAGE 5

2.4 HTTP REQUEST 6

2.5 HTTP RESPONSE 8

2.6 HTTP ENTITY 12

3. การเขียนโปรแกรมเพื่อทำการติดต่อบนระบบเครือข่าย 13

3.1 SOCKET API 13

3.2 การทำงานของโปรแกรมที่ติดต่อสื่อสารบนระบบเครือข่าย 15

3.3 ตัวแปรโครงสร้าง SOCKET ADDRESS (SOCKET ADDRESS STRUCTURE) 15

3.4 การสร้าง SOCKET เพื่อใช้ในการติดต่อสื่อสารโดยฟังก์ชัน SOCKET() 18

3.5 การกำหนด ADDRESS อ้างอิงสำหรับ SOCKET ด้วยฟังก์ชัน BIND() 19

3.6 การสร้างการเชื่อมต่อในโปรโตคอล TCP โดยใช้ฟังก์ชัน CONNECT(), LISTEN() และ ACCEPT() 20

3.7 การรับ/ส่งข้อมูลในโปรโตคอล TCP โดยใช้ฟังก์ชัน SEND() และ RECV() 22

3.8 การยกเลิกการเชื่อมต่อในโปรโตคอล TCP โดยใช้ฟังก์ชัน CLOSE() 23

3.9 การรับ/ส่งข้อมูลในโปรโตคอล UDP โดยใช้ฟังก์ชัน SENDTO() และ ฟังก์ชัน RECVFROM()....24
3.10 ตัวอย่างโปรแกรม TCP ECHO SERVER, ECHO CLIENT....25
3.11 ตัวอย่างโปรแกรม UDP ECHO SERVER, ECHO CLIENT....31
4. WWW PROXY....37
4.1 PROXY ทำหน้าที่เป็น GATEWAY....38
4.2 WEB CACHING....38
4.3 การทำงานของ WWW PROXY....40
5. การพัฒนา WWW PROXY ที่สามารถนำข้อมูล WWW ไปประมวลผล....44
5.1 การพัฒนาโปรแกรม....46
5.2 การทำงานของโปรแกรม WWW PROXY ในบริการหลายไคลเอนท์พร้อมกัน....52
5.3 การนำข้อมูล WWW ที่รับ/ส่งผ่านโปรแกรม WWW PROXY ไปประมวลผล....57
6. สรุปผลการศึกษา....61
6.1 โปรแกรม WWW PROXY และการใช้งาน....61
6.2 APPLICATION PROGRAM ตัวอย่าง : การแปลเอกสาร HTML ภาษาอังกฤษเป็นภาษาไทย....63
6.3 สรุปผลการศึกษา....65
บรรณานุกรม....66
ภาคผนวก....67

บทที่ 1.
บทนำ

ในปัจจุบัน หลังจากที่มีการเกิดขึ้นของเครือข่าย Internet บริการ World-Wide Web ก็ได้รับความนิยมอย่างสูง และนับได้ว่าเข้ามามีบทบาททั้งทางด้านข้อมูลข่าวสารและความบันเทิงต่างๆ มากมาย เนื่องจากเป็นแหล่งข้อมูลขนาดใหญ่ และแสดงข้อมูลในรูปแบบของรูปภาพ, เสียง, ตัวอักษร ฯลฯ ซึ่งทำให้การเข้าถึงข้อมูลและการทำความเข้าใจสามารถทำได้ง่าย และสามารถนำไปใช้ได้รวดเร็ว อีกทั้งยังมีรูปแบบที่สวยงาม ซึ่งเป็นผลทำให้มีความต้องการใช้งานมากขึ้น ส่งผลให้ประสิทธิภาพการใช้งานระบบโดยรวมลดต่ำลง จึงได้มีการพัฒนาโปรแกรม WWW proxy ขึ้นเพื่อช่วยในการใช้บริการ world wide web ที่มีความรวดเร็วมากขึ้น

โปรแกรม WWW proxy เป็นโปรแกรมที่พัฒนาขึ้นเพื่อช่วยแก้ไขปัญหาความคับคั่งในระบบเครือข่าย อันเนื่องมาจากความต้องการใช้บริการ world wide web ที่มีมากขึ้น อาศัยเทคนิควิธีที่เรียกว่า proxy caching โดยใช้เครื่องคอมพิวเตอร์ 1 เครื่องหรือมากกว่า ทำงานเป็นเสมือน cache ของทรัพยากรใน World-Wide Web ให้กับ www client ที่ต้องการทรัพยากรเหล่านั้น เพื่อช่วยให้การเรียกใช้และเข้าถึงข้อมูลต่างๆที่เคยมีการเรียกใช้มาแล้ว สามารถทำโดยดึงข้อมูลจาก cache ได้ทันที เป็นการช่วยลดการใช้งานเครือข่ายและการเข้าถึงเครื่องเซิร์ฟเวอร์ผู้ให้บริการ World-Wide Web อีกด้วย

ในปัจจุบันนั้น วัตถุประสงค์หลักของการนำเอา WWW proxy มาใช้งาน คือช่วยให้การเข้าถึงข้อมูล WWW รวดเร็วมากขึ้น โดยอาศัยหลักการของ web caching ทั้งนี้ข้อมูลที่ได้รับนั้น นำไปแสดงผลผ่านทาง web browser โดยตรงเท่านั้น ในขณะที่เราอาจสามารถใช้ความสามารถของ WWW proxy โดยนำข้อมูลต่างๆที่รับ/ส่งผ่านโปรแกรม WWW proxy ไปประมวลผลเพื่อประโยชน์อื่นๆโดยก่อนการส่งไปแสดงผลได้ ในบทความนี้จะทำการศึกษาถึงวิธี, ความเป็นไปได้ และการพัฒนาโปรแกรม web proxy เพื่อนำข้อมูลไปประมวลผลโดย application อื่นๆ นอกเหนือจากการแสดงผลโดย web browser เท่านั้น

1.1 วัตถุประสงค์

1. เพื่อศึกษาหลักการและการทำงานของโปรแกรม WWW proxy ที่ใช้งานอยู่ในปัจจุบัน
2. ศึกษาถึงแนวทางในการพัฒนาโปรแกรม WWW proxy เพื่อสามารถนำข้อมูลเอกสารที่มีการรับ/ส่งผ่านทางโปรแกรม WWW proxy ไปใช้ประโยชน์โดยการประมวลผลอื่นๆ เช่น การแปลภาษาอังกฤษให้เป็นภาษาไทย เป็นต้น ก่อนการส่งผลลัพธ์ไปยัง WWW browser เพื่อไปแสดงผล
3. ทำการพัฒนาโปรแกรมต้นแบบเพื่อทดสอบความเป็นไปได้ในการนำข้อมูลดังกล่าวไปประมวลผล และการใช้งานโปรแกรม

1.2 ขอบเขตของโครงการ

- ทำการศึกษาความเป็นไปได้ในการปรับปรุงโปรแกรม WWW proxy เพื่อให้สามารถทำงานส่งข้อมูลไปประมลผลยัง application program ก่อนแสดงผลยัง WWW browser
- ทำการพัฒนาโปรแกรม WWW proxy ที่สามารถทำงานส่งข้อมูลไปประมวลผลยัง application program ก่อนแสดงผลยัง WWW browser ซึ่งมีคุณสมบัติสามารถสนับสนุนข้อมูลเอกสาร html พื้นฐานได้
- โปรแกรม WWW proxy ที่ทำการศึกษาและพัฒนานี้ จะไม่รวมถึงคุณสมบัติและวิธีการในการ caching ข้อมูลโดยโปรแกรม WWW proxy

1.3 ผลที่คาดว่าจะได้รับ

- พัฒนาโปรแกรม WWW proxy ที่สามารถทำงานส่งข้อมูลไปประมลผลยัง application program ก่อนแสดงผลยัง WWW browser ซึ่งมีคุณสมบัติสามารถสนับสนุนข้อมูลเอกสาร html พื้นฐาน ได้
- สามารถนำแนวคิดในการปรับปรุงโปรแกรม WWW proxy เพื่อให้สามารถทำงานส่งข้อมูลไปประมลผลยัง application program ก่อนแสดงผลยัง WWW browser ไปใช้ในการพัฒนาต่อไปได้

1.4 เนื้อหาของบทความ

เนื้อหาสำหรับโครงการพัฒนาโปรแกรมในบทแรกนี้จะกล่าวถึงความเป็นมา ความสำคัญและวัตถุประสงค์ในการทำโครงการพัฒนาโปรแกรมนี้ บทที่ 2 จะกล่าวถึงหลักการ World Wide Web และ Hypertext Transfer Protocol ได้แก่รูปแบบของ HTTP message ต่างๆ ที่ใช้สื่อสารกันใน

บริการ WWW บทที่ 3 กล่าวถึงการเขียนโปรแกรมเพื่อทำงานติดต่อสื่อสารบนเครือข่ายอินเตอร์เน็ท โดยจะกล่าวถึง socket API พื้นฐานที่ใช้ในการเขียนโปรแกรม บทที่ 4 อธิบายถึงการทำงานของโปรแกรม WWW proxy และขั้นตอนการทำงานต่างๆ บทที่ 5 การพัฒนาโปรแกรมเพื่อทำการทดสอบ และบทที่ 6 บทสรุปและวิธีการใช้งานโปรแกรม

บทที่ 2

World Wide Web และ Hyper Text Transfer Protocol

2.1 World Wide Web

World Wide Web เป็นบริการหนึ่งที่เกิดขึ้นบนเครือข่ายอินเตอร์เน็ท ที่สามารถให้บริการข้อมูลข่าวสารต่างๆที่อยู่ในรูปแบบของ Hypertext และ Hypermedia (คือประกอบด้วยข้อมูลตัวอักษร, รูปภาพ, เสียง, ไฟล์ข้อมูลต่างๆ ฯลฯ โดยจะมี link ไปยังข้อมูลอื่นๆที่มีความเกี่ยวข้อง) อาศัยการรับ/ส่งข้อมูลโดย HTTP (Hyper Text Transfer Protocol)

World Wide Web ถูกพัฒนาขึ้นโดย CERN โดยในครั้งแรกนั้น เป็นเพียงการเชื่อมโยง (link) ระหว่างข้อมูลเอกสารหนึ่งไปยังข้อมูลเอกสารอื่นๆเท่านั้น และได้ถูกพัฒนาเรื่อยมาโดยมีการรวมเอาสื่อข้อมูลชนิดต่างๆเข้ามาไว้ใน WWW นอกเหนือจากข้อมูลเอกสารธรรมดา โดยจะมีลักษณะเป็นข้อมูลที่มีการเชื่อมโยงไปยังข้อมูลอื่นๆภายในเอกสารเดียวกันหรือเอกสารอื่นๆ ทำให้การขยายตัวของ WWW เป็นไปอย่างรวดเร็ว และได้รับความนิยมอย่างสูง

2.2 Hyper Text Transfer Protocol (HTTP)

Hyper Text Transfer Protocol (HTTP) เป็นโปรโตคอลในระดับ application ที่ใช้ในการรับ/ส่ง และกระจายทรัพยากรและข้อมูลชนิดต่างๆ (hypermedia) โดยอาศัยหลักการของ request – response เป็นหลัก HTTP ถูกนำไปใช้สำหรับบริการ World Wide Web จนเป็นที่แพร่หลาย โดยในการทำงานนั้นจะประกอบด้วย client ทำการร้องขอข้อมูลต่างๆ (request) โดยอาศัย TCP connection ไปยัง server ที่ให้บริการที่หมายเลข port 80 (ถ้าไม่มีการเปลี่ยนแปลงเป็นอย่างอื่น) ซึ่งในส่วนมากจะเริ่มต้นจาก client (user agent เช่น browsers, editors, spiders ฯลฯ) เป็นผู้สร้าง request ส่งไปยัง server โดยตรง ผ่านทางการเชื่อมต่อ (connection) ระหว่าง client และ server เพียง connection เดียวสำหรับแต่ละทรัพยากรที่ร้องขอ จากนั้น server จะทำการตรวจสอบและตอบกลับ (response) ไปยัง client พร้อมทั้งบอกสถานะและส่งทรัพยากรที่ร้องขอ (ถ้ามี) กลับไปยัง client และทำการปิดการเชื่อมต่อนั้น

2.3 HTTP Message

HTTP Message คือข้อความที่ส่ง/รับกันสำหรับทั้ง HTTP request และ HTTP response โดยสามารถแยกออกได้เป็น Simple-Request, Simple-Response สำหรับ HTTP/0.9 และ Full-Request, Full-Response สำหรับ HTTP/1.0 ขึ้นไป ซึ่งสามารถแสดงในรูปของ BNF ได้ดังนี้

```
Simple-Request      = "GET" SP Request-URI CRLF
Simple-Response     = [Entity-Body]
Full-Request        = Request-Header
                      *( General-Header | Request-Header | Entity-Header )
                      CRLF
                      [Entity-Body]
Full-Response       = Status-Line
                      *( General-Header | Response-Header | Entity-Header )
                      CRLF
                      [Entity-Body]
SP                  = <US-ASCII SP, space (32)>
CRLF                = CR LF
CR                  = <US-ASCII CR, return (13)>
LF                  = <US-ASCII LF, linefeed (10)>
[Entity-Body]       = ตัวข้อมูลที่ส่งไปใน message
```

2.3.1 Message Header

รูปแบบของ HTTP header ต่างๆ (ทั้ง General-Header, Request-Header, Response-Header และ Entity-Header) (ตามรูปแบบใน RFC 822) ประกอบด้วย ชื่อฟิลด์ (filed name) ตามด้วย ":" และ ค่าของฟิลด์นั้นๆ (filed value)

```
HTTP-Header         = field-name ":" [field-value] CRLF
```

2.3.2 General Header

General Header เป็น header ที่ใช้ในการระบุข้อมูลทั่วไปสำหรับทั้ง request message และ response message โดยจะมีอยู่ในทุกๆ message ที่มีการรับ/ส่งกัน

```
General Header      = Date | Pragma
```

General Header ได้แก่

- Date เป็นฟิลด์ที่ระบุวันเวลาของ HTTP message นั้นๆ โดยอยู่ในรูปแบบตาม RFC822, RFC850 และ ANSI C's asctime() format
- Pragma ใช้ในการกำหนดพฤติกรรมต่างๆที่ต้องการให้ผู้รับ HTTP message ทั้งหมดกระทำ โดยค่าที่เป็นไปได้ ได้แก่ "no-cache" หรือค่าอื่นๆที่อาจมีการระบุเพิ่มเติมในภายหลัง

2.4 HTTP Request

HTTP Request เป็น message ที่ client ที่ต้องการขอใช้ทรัพยากร ส่งไปยัง server เพื่อร้องขอทรัพยากรนั้นๆจาก server ที่ให้บริการ โดยจะประกอบด้วย Request-Line, วิธีการ (method) ที่ทำการร้องขอ, ระบุทรัพยากรที่ต้องการ และเวอร์ชั่นของโปรโตคอลที่ใช้

รูปแบบของ HTTP request สามารถแยกออกได้เป็น Simple-Request สำหรับ HTTP version 0.9 และ Full-Request สำหรับ HTTP version 1.0 เป็นต้นไป

Simple-Request = "GET" SP Request-URI CRLF

Full-Request = Request-Header
*(General-Header | Request-Header | Entity-Header)
CRLF
[Entity-Body]

2.4.1 Request Line

Request-Line ประกอบด้วย method (ได้แก่ "GET", "HEAD", "POST" หรืออื่นๆที่อาจมีเพิ่มเติม) ตามด้วย Request-URI และ protocol version

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

ตัวอย่างเช่น

GET http://www.w3.org/pub/www/TheProject.html HTTP/1.0

2.4.2 Request Header

Request-Header เป็นส่วนของข้อมูลเพิ่มเติมเกี่ยวกับ request และตัว client เอง ที่ client จะส่งไปให้กับ server

Request-Header = Accept
| Accept-Charset | Accept-Encoding | Accept-Language
| Authorization
| From

| Host

| If-Modified-Since | If-Unmodified-Since

| If-Match | If-None-Match

| If-Range

| Max-Forwards | Proxy-Authorization | Range

| Referer

| User-Agent

ประกอบด้วย field ต่างๆที่สำคัญ ดังนี้

- Accept ใช้สำหรับระบุชนิดของข้อมูลเฉพาะที่ client ต้องการรับจากการร้องขอ
- Authorization เป็นข้อมูลสำหรับ authority ต่างๆ
- From ข้อมูล Internet e-mail address ของผู้ใช้
- If-Modified-Since ใช้สำหรับเป็นเงื่อนไขในการ "GET" ข้อมูล โดยหากข้อมูลมีการเปลี่ยนแปลงแก้ไข ภายหลังจากวัน/เวลาที่ระบุใน filed นี้ server จึงจะส่งข้อมูลนั้นกลับไปให้กับ client มิฉะนั้น server จะส่งข้อความสถานะ 304 (not modified) แทน
- Referrer URL ที่อ้างอิงถึง และก่อให้เกิด request นี้
- User-Agent ข้อมูลเกี่ยวกับ user agent, โปรแกรมที่ใช้, version ของโปรแกรม

ตัวอย่าง HTTP Request เช่น

(1) GET http://www.w3.org/pub/www/TheProject.html HTTP/1.0

(2) Accept: image/gif, image/x-xbitmap, image/jpeg, image/pipeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*

(3) Accept-Language: th

(4) Accept-Encoding: gzip, deflate

(5) User-Agent: Mozilla/4.0 (compatible; MSIE 50; Windows 98; DigExt)

(6) Host: 161.200.154.200

(7)

จะเห็นว่า ในส่วนของ header ต่างๆของ HTTP request นั้น อาจมีหรือไม่มีข้อมูลในฟิลด์ใดๆก็ได้ ทั้งนี้ขึ้นอยู่กับความเหมาะสมและความจำเป็นในการใช้งานโดยจากตัวอย่างสามารถอธิบายได้ดังนี้

บรรทัดที่ (1) เป็นส่วนของ request line โดยเป็นการข้อใช้ทรัพยากร (แฟ้มเอกสาร html ชื่อ TheProject.html) จาก WWW server " www.w3.org " โดยเอกสาร html นี้อยู่ใน path " /pub/www " โดยเป็นการขอใช้ทรัพยากรแบบ GET และทำการติดต่อโดยใช้ HTTP version 1.0

บรรทัดที่ (2)-(4) เป็นส่วนของ request header โดยเป็นการระบุชนิดของข้อมูลที่ client ยอมรับได้ รวมไปถึงภาษาและรูปแบบการเข้ารหัสที่ client สามารถยอมรับได้

บรรทัดที่ (5)-(6) เป็นส่วนของ request header ที่ระบุข้อมูลต่างๆเกี่ยวกับ client โดยระบุว่า user agent ที่ใช้คือ โปรแกรม Microsoft Internet Explorer 5.0 และทำการติดต่อมาจากเครื่องคอมพิวเตอร์ที่มีหมายเลข IP address เป็น 161.200.154.200

ทั้งนี้ในส่วนของ Entity ที่ส่งมากับ HTTP request นั้น จะมีหรือไม่มีก็ได้ โดยจะเป็นข้อมูลเกี่ยวกับค่าพารามิเตอร์ต่างๆที่ client ต้องการส่งไปให้กับ WWW server เพื่อนำไปประมวลผล (เช่น ในการส่งค่าตัวแปรจากการ submit form ในหน้า web page เป็นต้น ซึ่งจะไม่กล่าวถึงในบทความนี้ หากต้องการรายละเอียดเพิ่มเติม สามารถค้นหาได้จากคำค้นเกี่ยวกับ CGI (Common Gateway Interface) และการพัฒนาเวปเพจ)

2.5 HTTP Response

HTTP response เป็น message ที่ server ส่งกลับไปยัง client หลังจากที่มีการร้องขอทรัพยากรมาจาก client และ server ทำการประมวลผลการร้องขอนั้นแล้ว โดยจะประกอบด้วยสถานะของการปะมวลผลการร้องขอ, header ต่างๆ และตัวผลลัพธ์จากการประมวลผลการร้องขอ (ถ้ามี)

รูปแบบของ HTTP response สามารถแยกออกได้เป็น Simple-Response สำหรับ http version 0.9 และ Full-Response สำหรับ HTTP version 1.0 เป็นต้นไป

```
Simple-Response     = [Entity-Body]
Full-Response       = Status-Line
                      *( General-Header | Response-Header | Entity-Header )
                        CRLF
                        [Entity-Body]
```

2.5.1 Status Line

Status Line เป็นส่วนของข้อความแสดงสถานะหลังการ server ทำการประมวลผลการร้องขอของ client แล้ว โดยจะมีเฉพาะใน message แบบ Full-Response เท่านั้น ประกอบด้วยเวอร์ชันของโปรโตคอล และหมายเลขแสดงสถานะ

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

ตัวอย่างเช่น

HTTP/1.0 200 OK

2.5.2 Status Code และ Reason Phrase

Status Code ประกอบด้วยตัวเลข 3 หลักใช้ในการระบุสถานะการทำงานตามที่มีการร้องขอจาก client ซึ่งจะเป็นส่วนหนึ่งใน Status Line โดยที่ Status Code แต่ละตัวจะมี Reason Phrase ใช้เป็นส่วนขยายให้สามารถอ่านเข้าใจได้ง่ายโดยมนุษย์

Status Code สามารถแบ่งได้ออกเป็น 5 กลุ่มใหญ่ๆดังนี้

- 1XX : Information เป็นกลุ่มของ Status Code ที่ขึ้นต้นด้วยตัวเลข "1" ใช้แสดงสถานะทั่วไป เช่น "ได้รับ request message แล้ว จะทำการประมวลผลต่อไป" (Request received, continuing process)
- 2XX : Success การประมวลผลกระทำได้สำเร็จเสร็จสิ้น และสมบูรณ์
- 3XX : Redirection จะต้องมีการกระทำอย่างใดอย่างหนึ่งก่อนจึงจะทำให้การประมวลผลการร้องขอสำเร็จ เช่น การทำ redirect ไปยัง URI อื่น
- 4XX : Client Error เกิดความผิดพลาดใน request message เช่น ไม่เป็นไปตามรูปแบบที่กำหนด หรือไม่สามารถตอบสนองต่อการร้องขอนั้นได้
- 5XX : Server Error เกิดความผิดพลาดที่ server ไม่สามารถประมวลผลต่อการร้องขอ (ที่ถูกต้อง)ได้

Status Code และ Reason Phrase ต่างๆ ได้แก่

Status-Code = "100" ; Continue | "101" ; Switching Protocols
| "200" ; OK | "201" ; Created
| "202" ; Accepted | "203" ; Non-Authoritative Information
| "204" ; No Content | "205" ; Reset Content
| "206" ; Partial Content | "300" ; Multiple Choices
| "301" ; Moved Permanently
| "302" ; Moved Temporarily
| "303" ; See Other | "304" ; Not Modified
| "305" ; Use Proxy | "400" ; Bad Request
| "401" ; Unauthorized | "402" ; Payment Required

| "403" ; Forbidden | "404" ; Not Found

| "405" ; Method Not Allowed

| "406" ; Not Acceptable | "407" ; Proxy Authentication Required

| "408" ; Request Time-out

| "409" ; Conflict | "410" ; Gone

| "411" ; Length Required | "412" ; Precondition Failed

| "413" ; Request Entity Too Large

| "414" ; Request-URI Too Large | "415" ; Unsupported Media Type

| "500" ; Internal Server Error | "501" ; Not Implemented

| "502" ; Bad Gateway | "503" ; Service Unavailable

| "504" ; Gateway Time-out | "505" ; HTTP Version not supported

| อื่นๆ ถ้ามีการกำหนดเพิ่มในภายหลัง

2.5.3 Response Header

Response Header เป็นส่วนของข้อมูลเพิ่มเติมต่างๆเกี่ยวกับ response ที่ server ต้องการจะส่งไปยัง client แต่ไม่สามารถระบุใน Status-Line ได้

Response-Header = Age

| Location

| Proxy-Authenticate | Public

| Retry-After

| Server

| Vary | Warning

| WWW-Authenticate

ประกอบด้วย field ต่างๆที่สำคัญดังนี้

- Age เป็นช่วงเวลาที่ server คาดว่า response นั้นๆจะมีอายุการใช้งานอยู่ (ก่อนที่จะมีการแก้ไข) โดยมีหน่วยเป็นวินาที
- Location แสดงที่อยู่ที่แท้จริงของทรัพยากรที่ร้องขอ (กรณีที่ต้องทำการ redirect)
- Server แสดงข้อมูลเกี่ยวกับ server, โปรแกรมที่ใช้, version ของโปรแกรม
- WWW-Authenticate ข้อมูลสำหรับ authority ต่างๆ

ตัวอย่าง HTTP Response เช่น

(1) HTTP/1.0 200 OK

(2) Server: Microsoft-IIS/4.0

(3) Date: Mon, 04 Sep 2000 04:22:58 GMT

(4) Content-Type: text/html

(5) Accept-Ranges: bytes

(6) Last-Modified: Sun 02 Jul 2000 03:46:24 GMT

(7) Content-Length: 2064

(8)

(9) <html>

...

...

(10) </html>

(11)

จากตัวอย่างเป็น HTTP response ที่ส่งมาจาก WWW server โดยสามารถแยกออกเป็นส่วนของ header ต่างๆ (บรรทัดที่ (1) ถึง (7)) และส่วนของข้อมูลทรัพยากรที่ client ร้องขอ (ส่วนของ Entity บรรทัดที่ (9) เป็นต้นไป) โดยเช่นเดียวกับ HTTP request จะเห็นว่า ในส่วนของ header ต่างๆของ HTTP response อาจมีหรือไม่มีข้อมูลในฟิลด์ใดๆก็ได้

บรรทัดที่ (1) เป็น status line ระบุ HTTP version ที่ WWW server ใช้ในการติดต่อ (ในที่นี้คือ HTTP version 1.0) และระบุสถานะต่างๆในรูปของ Status Code และ Reason Phrase "200 OK"

บรรทัดที่ (2) – (7) เป็นส่วนของ Response header โดยมีการระบุโปรแกรม WWW server ที่ใช้ คือโปรแกรม Microsoft-IIS เวอร์ชัน 4.0 (บรรทัดที่ (2)), วันที่และเวลาของข้อความ response นี้ (บรรทัดที่ (3)), ชนิดของข้อมูลที่ส่งมาในส่วนของ Entity (บรรทัดที่ (4)), หน่วยของความยาวของ Entity (บรรทัดที่ (5)), วันที่และเวลาที่มีการแก้ไขล่าสุด (บรรทัดที่ (6)) และความยาวทั้งหมดของข้อมูลที่ส่งมาในส่วนของ Entity (บรรทัดที่ (7))

บรรทัดที่ (9) – (11) เป็นส่วนของ HTTP Entity ซึ่งบรรจุข้อมูลทรัพยากรตามที่ client ร้องขอ โดยในส่วนนี้อาจไม่มีก็ได้ขึ้นอยู่กับลักษณะการทำงานและสถานะการประมวลผลตามที่ client ร้องขอ

2.6 HTTP Entity

HTTP Entity เป็นส่วนของข้อมูลที่ส่งใน Full Request หรือ Full Response โดยจะประกอบด้วย Entity-Header และ Entity-Body

2.6.1 Entity Header

Entity Header เป็นส่วนที่ระบุข้อมูลเพิ่มเติมต่างๆเกี่ยวกับข้อมูลที่ส่งมาใน request หรือ response นั้นๆ

Entity-Header = Allow
| Content-Encoding
| Content-Length
| Content-Type
| Expires
| Last-Modified
| อื่นๆถ้ามีการนิยามเพิ่มเติมในภายหลัง

ประกอบด้วย field ต่างๆที่สำคัญ ดังนี้

- Allow ระบุ method ที่สนับสนุนให้มีการใช้ได้ในการร้องขอทรัพยากรตามที่ระบุใน Request-URI
- Content-Encoding ใช้ในการระบุชนิดของการเข้ารหัส (encoding) ข้อมูลใน Entity Body เพื่อให้ผู้รับสามารถทำการถอดรหัส (decode) ได้อย่างถูกต้อง เช่น x-gzip เป็นต้น
- Content-Length ระบุความยาวของข้อมูล (Entity Body) เป็นจำนวน Byte
- Content-Type เป็นส่วนที่ระบุชนิดของสื่อ (media type) ของข้อมูลใน Entity Body
- Expires วันที่และเวลาที่ข้อมูลใน Entity Body จะหมดอายุการใช้งานก่อนที่จะถูกแก้ไข ซึ่งสามารถใช้ในการตัดสินใจในการทำ caching ได้
- Last-Modified วันที่และเวลาที่ผู้ส่งคาดว่าข้อมูลใน Entity Body ได้ถูกแก้ไขครั้งล่าสุด

2.6.2 Entity Body

เป็นส่วนของข้อมูลทรัพยากรที่มีการส่ง/รับกันโดย request message และ response message โดยอาจจะมีหรือไม่มีก็ได้ ขึ้นกับชนิดของ message และสถานะการทำงาน ทั้งนี้รูปแบบของข้อมูลภายใน Entity Body จะเป็นไปตามที่ระบุใน Entity Header ต่างๆที่กล่าวมาแล้ว

บทที่ 3
การเขียนโปรแกรมเพื่อทำการติดต่อสื่อสารบนระบบเครือข่าย

ในการทำงานของโปรแกรมที่ทำงานติดต่อสื่อสารกันในระบบเครือข่าย (เช่น Internet) จะต้องอาศัยข้อตกลง (protocol) ต่างๆทั้งทางด้านรูปแบบและขั้นตอนในการรับ/ส่งข้อมูล รวมไปถึงรูปแบบของตัวข้อมูลที่รับ/ส่งกันเองด้วย เพื่อให้โปรแกรมต่างๆทั้งทางด้าน client และ server สามารถสื่อสารกันได้ ภายใต้สภาพแวดล้อมที่แตกต่างกันออกไปอย่างมากมายในระบบเครือข่าย

การติดต่อสื่อสารกันในเครือข่าย Internet นั้น จะอาศัย TCP/IP protocol suit เป็นข้อตกลงในการติดต่อสื่อสาร โดยการสื่อสารกันนั้นสามารถทำได้ 2 ลักษณะคือแบบที่มีการสร้างการเชื่อมต่อ (connection) ไว้ก่อนที่จะมีการติดต่อสื่อสาร (Connection-oriented) และแบบที่ไม่มีการสร้างการเชื่อมต่อไว้ (Connectionless)

3.1 Socket API

Socket API เป็น Application Programming Interface ที่ได้ถูกพัฒนาขึ้นโดย University of California โดยเป็นส่วนหนึ่งอยู่ใน ระบบ UNIX ของ BSD 4.2 ในปี ค.ศ. 1983 เพื่อใช้ในการพัฒนาโปรแกรมที่ต้องการติดต่อสื่อสารในเครือข่ายอินเตอร์เน็ท โดยใช้โปรโตคอล TCP/IP ในการติดต่อสื่อสาร และได้ถูกพัฒนาและปรับปรุงไปหลายรูปแบบตามแต่ละระบบปฏิบัติการ หรือภาษาที่ใช้ในการพัฒนาโปรแกรม แต่จะคงลักษณะการทำงานเหมือนเดิมไว้ ทั้งนี้ในบทนี้จะกล่าวถึง socket API มาตรฐาน ซึ่งถูกสร้างขึ้นโดย University of California (Berkeley BSD-socket) ซึ่งเป็นพื้นฐานที่ถูกนำไปปรับปรุงเป็น socket API อื่นๆ (เช่น windows socket เป็นต้น)

ในการเขียน program เพื่อทำการติดต่อสื่อสารบนระบบเครือข่ายอินเตอร์เน็ทโดยใช้ socket API นั้น แต่ละโปรเซสที่ต้องการติดต่อสื่อสารกับโปรเซสอื่น จะต้องสร้าง socket ขึ้นมาเพื่อเป็นช่องทางในการส่ง/รับข้อมูล ซึ่งในความเป็นจริงแล้ว socket ก็คือ ช่องทางที่ใช้ในการติดต่อระหว่างโปรแกรมและระบบปฏิบัติการ เพื่อให้ระบบปฏิบัติการจัดการส่งข้อมูลเหล่านั้นไปในเครือข่ายโดยผ่านทางอุปกรณ์เครือข่าย (network device) ต่อไป

ชนิดของ socket นั้น มีหลายชนิดขึ้นกับลักษณะของการรับส่งข้อมูล ที่นิยมได้แก่

- **Stream socket** (SOCK_STREAM) ใช้สำหรับการรับ/ส่งข้อมูลด้วย protocol TCP ใน Transport Layer ซึ่งในการติดต่อจะต้องมีการสร้างการเชื่อมต่อ (connection) ระหว่าง

กันทั้ง 2 ฝ่าย (connection-oriented) ตามกรรมวิธี three-way handshake ของ protocol TCP และมีการตรวจสอบความถูกต้องของข้อมูล ซึ่งจะทำให้การรับ/ส่งข้อมูลมีความน่าเชื่อถือมากขึ้น

- **Datagram socket** (SOCK_DGRAM) ใช้สำหรับการรับ/ส่งข้อมูลด้วย protocol UDP โดยที่การติดต่อนั้น ไม่ต้องมีการสร้างการเชื่อมต่อระหว่างทั้ง 2 ฝ่าย (connectionless) และไม่มีการตรวจสอบความถูกต้องของการรับ/ส่งข้อมูล
- **Sequenced-packet socket** (SOCKET_SEQPACKET) ใช้สำหรับการรับ/ส่งข้อมูลด้วย protocol IPX/SPX

3.1.1 การติดต่อสื่อสารโดยใช้ stream socket (TCP socket)

การทำงานของ Stream socket อาศัย TCP (Transport Control Protocol) โดยในการติดต่อสื่อสารนั้น จะต้องมีการสร้างช่องทางการเชื่อมต่อ (connection) ระหว่าง server และ client ไว้ก่อนแล้วจึงทำการส่ง/รับข้อมูลผ่านทางการเชื่อมต่อนั้น ซึ่งจะมีคุณสมบัติดังนี้

- เป็น connection–oriented นั่นคือในการสื่อสารกันนั้น จะต้องสร้างการเชื่อมต่อระหว่างกันเสียก่อน (ตามวิธี Three-way handshake) การเชื่อมต่อนั้นจะระบุถึง process ปลายทางโดยอาศัย IP address ของเครื่องปลายทาง และหมายเลข Port

- ส่งข้อมูลผ่านทาง connection ที่สร้างขึ้น ในรูปแบบของ Byte Stream (มีความยาวไม่จำกัด) ซึ่งอาจมีการแบ่งออกเป็น packet ย่อยๆตามความเหมาะสม และเมื่อผู้รับได้รับข้อมูล packet นั้นๆ แล้ว จะมีการส่ง packet ตอบกลับ (Acknowledge) ไปยังผู้ส่ง ทำให้การสื่อสารมีความน่าเชื่อถือมากขึ้น

ข้อมูลที่รับ/ส่งกันโดยใช้ stream socket จะมีความถูกต้อง, เป็นไปตามลำดับก่อนหลัง และไม่มีความซ้ำซ้อนของข้อมูล เนื่องมาจากลักษณะการทำงานและคุณสมบัติของ Protocol TCP

3.1.2 การติดต่อสื่อสารโดยใช้ Datagram socket (UDP Socket)

การทำงานของ Datagram socket อาศัย UDP (User Data Protocol) โดยในการติดต่อสื่อสารนั้น สามารถทำการส่งข้อมูลไปยังผู้รับได้ทันที โดยไม่ต้องสร้างช่องทางการเชื่อมต่อระหว่างผู้ส่งและผู้รับไว้ก่อน ซึ่งมีคุณสมบัติดังนี้

- เป็นการสื่อสารแบบ connectionless นั่นคือในการสื่อสารกันนั้น ไม่ต้องสร้าง connection ระหว่างผู้รับและผู้ส่งข้อมูล โดยผู้ส่งสามารถส่งข้อมูลนั้นๆ ไปยังผู้รับได้ทันที โดยจะระบุถึง โปร-

เซสซึ่งเป็นผู้รับปลายทางโดยอาศัย IP address ของเครื่องปลายทาง และหมายเลข Port ที่ โปรเซสปลายทางนั้นใช้ในการติดต่อสื่อสาร

- ส่งข้อมูลในรูปแบบของ datagram คือเป็นข้อมูลที่มีความยาวจำกัด ไปยังเครื่องปลายทางโดยที่ไม่มีการสร้าง connection ก่อน และ ไม่มีการตอบกลับจากผู้รับเพื่อยืนยันความถูกต้องของการส่ง/รับข้อมูล ดังนั้น การสื่อสารโดยใช้ Datagram socket จึงเป็นแบบ non-reliable

- ข้อมูลที่รับ/ส่งกันโดยใช้ datagram socket จะไม่รับประกันความถูกต้อง ทั้งนี้ข้อมูลที่ผู้รับได้รับอาจไม่เรียงไปตามลำดับตามที่ผู้ส่งได้ส่งข้อมูลไป และอาจมีความซ้ำซ้อนของข้อมูลเกิดขึ้น เนื่องมาจากลักษณะการทำงานและคุณสมบัติของ UDP (User Datagram Protocol)

3.2 การทำงานของโปรแกรมที่ติดต่อสื่อสารบนระบบเครือข่าย

ในการเขียนโปรแกรมเพื่อทำการติดต่อรับ/ส่งข้อมูลกันระหว่างโปรแกรมบนระบบเครือข่ายนั้น สามารถแยกหน้าที่หลักๆของโปรแกรมได้เป็น 2 ลักษณะคือโปรเซสที่ทำหน้าที่เป็น server และโปรเซสที่ทำหน้าที่เป็น client โดยมีลักษณะพอสังเขปดังนี้

server จะต้องทำการสร้าง network I/O (socket) เพื่อใช้ในการติดต่อ จากนั้น server จะเข้าสู่สถานะ listen เพื่อรอการขอเชื่อมต่อ (ในกรณี Stream socket) หรือรอข้อมูลที่ส่งมาจาก client (ในกรณี Datagram socket) เมื่อมีการร้องขอการเชื่อมต่อจาก client เข้ามายัง socket ที่สร้างไว้ server ก็จะพิจารณายอมรับการเชื่อมต่อนั้น (accept) และจะสามารถรับ/ส่งข้อมูลและนำข้อมูลเหล่านั้นไปประมวลผลต่อไป

client เมื่อได้สร้าง network I/O (socket) เพื่อใช้ในการติดต่อกับ server แล้ว จะสามารถส่ง/รับข้อมูลไปยัง server (ในกรณี Datagram socket) หรือ ขอการเชื่อมต่อกับ server (ในกรณี Stream socket) ได้ทันที และจะสามารถรับ/ส่งข้อมูลและนำข้อมูลเหล่านั้นไปประมวลผลต่อไป

3.3 ตัวแปรโครงสร้าง socket address (socket address structure)

socket address เป็นตัวแปรโครงสร้างที่ใช้ในการกำหนด address อ้างอิง และคุณสมบัติต่างๆเกี่ยวกับ socket เพื่อใช้ในการรับ/ส่งข้อมูลระหว่างโปรเซส ซึ่ง socket address มีหลายประเภทขึ้นกับชนิดของ protocol ที่ใช้ เช่น internet socket หรือ IP v4 socket address structure (sockaddr_in), IPv6 socket address structure (sockaddr_in6), IPX socket address structure (sockaddr_ipx) เป็นต้น โดยลักษณะของตัวแปรโครงสร้างเหล่านี้จะแตกต่างกันออกไปตามแต่ละชนิด ตัวอย่างเช่น

1. IPv4 socket address structure เป็นตัวแปรโครงสร้างสำหรับ socket ที่ใช้ในการติดต่อสื่อสารในเครือข่าย Internet โดยใช้ IP address version 4 โดยได้มีการระบุไว้ใน header file ของภาษา C คือ <netinet/in.h> มีโครงสร้างดังนี้

```
struct  in_addr {
        in_addr_t       s_addr;         /* 32-bit  IPv4 address*/
};
struct sockaddr_in {
        uint8_t         sin_len;        /* length of structure (16) */
        sa_family_t     sin_family;     /* AF_INET */
        in_port_t       sin_port        /* 16-bit TCP or UDP port number */
        struct in_addr  sin_addr        /* 32-bit IPv4 address */
        char            sin_zero[8];    /* unused */
}
```

รูปที่ 3.1 แสดง Internet (IPv4) socket address structure: sockaddr_in

2. IPv6 socket address structure เป็น ตัวแปรโครงสร้างสำหรับ socket ที่ใช้ในการติดต่อสื่อสารในเครือข่าย Internet โดยใช้ IP version6 โดยได้มีการระบุไว้ใน header file <netinet/in.h> มีโครงสร้างดังนี้

```
struct in6_addr {
        uint8_t         .s6_addr[16];   /* 128-bit IPv6 address */
};
#define SIN6_LEN                        /* required for compile-time tests */
```

```
struct sockaddr_in16 {
        uint8_t         sin6_len;       /* length of this struct [24] */
        sa_family_t     sin6_family;    /* AF_INET6 */
        in_port_t       sin6_port;      /* transport layer port# */
        uint32_t        sin6_flowinfo;  /* priority & flow label */
        struct in6_addr         sin6_addr;      /* IPv6 address */
};
```

รูปที่ 3.2 แสดง IPv6 socket address structure: sockaddr_in6

จะเห็นว่าโครงสร้างของ socket ชนิดต่างๆมีความแตกต่างกัน แต่ในการใช้งานนั้น จะต้องมีการส่งตัวแปรชนิดโครงสร้าง socket เหล่านี้เป็นพารามิเตอร์ไปให้กับฟังก์ชันต่างๆ ดังนั้นจึงมีการกำหนด ตัวแปรโครงสร้าง generic socket address structure ขึ้น เพื่อใช้เป็น generic pointer เพื่อให้สามารถส่งค่าตัวแปร socket ชนิดต่างๆเป็นพารามิเตอร์ไปยังฟังก์ชันเดียวกันได้

โครงสร้างตัวแปร generic socket ได้ถูกกำหนดไว้ใน header file <sys/socket.h> โดยมีโครงสร้างดังนี้

```
struct sockaddr {
        uint8_t         sa_len;
        sa_family_t     sa_family;      /* address family: AF_XXX value */
        char            sa_data[14];    /* protocol-specific address */
}
```

รูปที่ 3.3 แสดง generic socket address structure: sockaddr

- Length (sa_len) เป็นความยาวของตัวแปรโครงสร้าง
- Address family (sa_family) ใช้กำหนดชนิดของ protocol ที่ใช้ และเป็นตัวกำหนดรูปแบบของ address ที่จะใช้ในการติดต่อ ในรูปแบบของ AF_XXX ดังแสดงในรูปที่ 3.4
- Data (sa_data) เป็น Protocol specific address ระบุ address ของ process ที่จะติดต่อด้วย ซึ่งรูปแบบของ address นี้ ขึ้นกับชนิดของ protocol ที่ใช้ เช่น ใน internet socket address (AF_INET) จะใช้ IP v4 address และ port number เป็นตัวระบุ address

AF_INET	IPv4
AF_INET6	IPv6
AF_NS	Netware
AF_IPX	IPX/SPX

รูปที่ 3.4 แสดง address family ชนิดต่างๆ

3.4 การสร้าง socket เพื่อใช้ในการติดต่อสื่อสารโดยฟังก์ชัน socket()

ในการเขียนโปรแกรมโดยใช้ socket API การรับ/ส่งข้อมูลระหว่างโปรเซสจะทำผ่านทางช่องทางการสื่อสารที่เรียกว่า socket ดังนั้น ก่อนที่จะทำการรับ/ส่งข้อมูลได้นั้น จะต้องสร้างช่องทางการสื่อสารนี้ขึ้นก่อน ซึ่งสามารถทำได้โดยเรียกใช้ฟังก์ชัน socket() โดยระบุชนิดของการสื่อสารและโปรโตคอลที่ต้องการใช้ (เช่น ใช้โปรโตคอล TCP และ IPv4, ใช้โปรโตคอล UDP และ IPv6 เป็นต้น) โดยมีรูปแบบดังนี้

```
#include <sys/socket.h>
int  socket(int family,  int type,  int protocol);
```

ฟังก์ชัน socket() จะทำการสร้าง socket ขึ้นและคืนค่า socket handler เพื่อใช้ในการระบุถึง socket ที่สร้างขึ้นเมื่อเวลาที่ใช้งาน

family เป็น family ของ socket ที่ต้องการสร้างขึ้น โดยจะมีค่าได้ตามที่ระบุในรูปที่ 3.4

type เป็นชนิดของโปรโตคอลที่ใช้ โดยจะมีค่าได้เป็น

- SOCK_STREAM สำหรับสร้าง stream socket (ใช้โปรโตคอล TCP ในการสื่อสาร)
- SOCK_DGRAM สำหรับสร้าง datagram socket (ใช้โปรโตคอล UDP ในการสื่อสาร)

protocol เป็นโปรโตคอลที่ใช้ซึ่งจะต้องสอดคล้องกับ *family*

3.5 การกำหนด address อ้างอิงสำหรับ socket ด้วยฟังก์ชัน bind()

ฟังก์ชัน bind() เป็นฟังก์ชันสำหรับการกำหนดค่า address อ้างอิงในการติดต่อซึ่งได้แก่ IP address และหมายเลข port ให้แก่ socket ทั้งแบบ stream socket และ datagram socket โดยมีรูปแบบดังนี้

```
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr, socklen_t addlen);
```

ฟังก์ชัน bind() ทำการกำหนด address อ้างอิงตามที่ระบุใน **addr* ให้กับ socket *sockfd* และทำการคืนค่า 0 เมื่อสามารถกำหนด address อ้างอิงได้สำเร็จ หรือ ค่า error ตามแต่ชนิดความผิดพลาดที่เกิดขึ้น

sockfd เป็น socket handler สำหรับ socket (ที่ได้สร้างขึ้นด้วยฟังก์ชัน socket()) ที่ต้องการกำหนด address อ้างอิง

addr เป็นตัวแปร pointer ไปยังตัวแปรโครงสร้าง socket address เพื่อเป็นตัวระบุ address อ้างอิงที่จะกำหนดให้กับ socket *sockfd*

addlen เป็นค่าความยาว (ขนาด) ของตัวแปรโครงสร้าง *addr*

3.6 การสร้างการเชื่อมต่อในโปรโตคอล TCP โดยใช้ฟังก์ชัน connect(), listen() และ accept()

ดังที่ได้กล่าวแล้วว่าในการติดต่อสื่อสารโดยใช้โปรโตคอล TCP สามารถกระทำได้โดยการสร้าง socket สำหรับ stream socket และการรับ/ส่งข้อมูลนั้นจะต้องมีการสร้างการเชื่อมต่อ (connection) ไว้ก่อนจึงจะกระทำได้ โดยสามารถแยกโปรแกรมที่ทำการติดต่อสื่อสารกันได้เป็น 2 ฝ่าย คือ ฝ่ายที่ร้องขอการสร้างการเชื่อมต่อ (ฝ่ายที่ทำ active open ตามกรรมวิธี 3-way handshake ของโปรโตคอล TCP) ในที่นี้จะเรียกว่า TCP client และฝ่ายที่รอรับการร้องขอสร้างการเชื่อมต่อ (ฝ่ายที่ทำ passive open ตามกรรมวิธี 3-way handshake ของโปรโตคอล TCP) ในที่นี้จะเรียกว่า TCP server

หลังจากที่ client และ server สร้าง socket เพื่อใช้ในการสื่อสารไว้แล้ว server จะต้องกำหนด address อ้างอิง (bind) ให้กับ socket ที่สร้างขึ้น เพื่อให้ client สามารถอ้างอิงถึง address ที่จะใช้ในการติดต่อไปยัง server ได้ หลังจากนั้น server จะทำการรอรับการร้องขอสร้างการเชื่อมต่อจาก โดยฟังก์ชัน listen() จนกว่าจะมี client ร้องขอสร้างการเชื่อมต่อโดยฟังก์ชัน connect() มายัง server ตาม address อ้างอิงที่ server ได้ระบุไว้ จากนั้น server จึงยอมรับการเชื่อมต่อโดยฟังก์ชัน accept() และทำการสร้างการเชื่อมต่อ (TCP connection) และสามารถรับ/ส่งข้อมูลระหว่าง client และ server ได้ต่อไป

ฟังก์ชัน listen()

ฟังก์ชัน listen() จะถูกเรียกใช้โดย TCP server และจะทำให้ socket ที่ระบุ อยู่ในสถานะ passive นั่นคือสามารถรับการเชื่อมต่อเมื่อมี client ร้องขอสร้างการเชื่อมต่อ (connect)

ฟังก์ชัน listen() มีรูปแบบดังนี้

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

sockfd เป็น socket handler ของ socket (ที่ได้สร้างขึ้นโดยฟังก์ชัน socket()) ที่ต้องการใช้ในรอรับการร้องขอการเชื่อมต่อ

backlog เป็นขนาดสูงสุดที่ยอมรับได้ของคิว (queue) ของการร้องขอที่รอการเชื่อมต่ออยู่

ฟังก์ชัน connect()

ฟังก์ชัน connect() จะถูกเรียกใช้โดย TCP client เพื่อร้องขอสร้างการเชื่อมต่อระหว่าง socket ที่ client ได้สร้างไว้ กับ socket ที่ server สร้างขึ้นและรอรับการเชื่อมต่อโดยฟังก์ชัน listen()

ฟังก์ชัน connect() มีรูปแบบดังนี้

```
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *servaddr, socklen_t len);
```

sockfd เป็น socket handler สำหรับ socket ที่ client สร้างขึ้นและต้องการใช้ในการเชื่อมต่อกับ server

servaddr เป็นตัวแปรโครงสร้าง socket address ที่ใช้ในการระบุ address อ้างอิง เพื่อใช้อ้างอิงในการขอสร้างการเชื่อมต่อไปยัง socket ของ server ที่ต้องการ ทั้งนี้ค่าของ servaddr นี้จะเป็นค่าเดียวกับค่า address อ้างอิงที่ server ได้กำหนดไว้โดยฟังก์ชัน bind()

len เป็นความยาวของตัวแปรโครงสร้าง *servaddr*

ฟังก์ชัน accept()

ฟังก์ชัน accept() จะถูกเรียกใช้โดย TCP server เมื่อมีการร้องขอสร้างการเชื่อมต่อ (connect) จาก client เข้ามายัง socket ที่ server ได้สร้างขึ้นและรอรับการเชื่อมต่ออยู่ (listen) และจะทำการสร้างการเชื่อมต่อ (TCP connection) ระหว่าง server และ client เพื่อใช้ในการรับ/ส่งข้อมูลต่อไป

ทั้งนี้ฟังก์ชัน accept() จะทำการสร้าง socket ใหม่และทำการเชื่อมต่อกับ client โดยใช้ socket ที่สร้างขึ้นมาใหม่นี้ และคืนค่ากลับเป็น socket handler ที่สร้างขึ้นมาใหม่นั้น โดย server สามารถนำ socket ที่สร้างขึ้นใหม่นี้ไปใช้ในการรับ/ส่งข้อมูลกับ client ได้

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *len);
```

sockfd เป็น socket handler สำหรับ socket ที่รอรับการเชื่อมต่ออยู่และมีการร้องขอสร้างการเชื่อมต่อจาก client เข้ามา

cliaddr เป็น pointer ไปยังตัวแปรโครงสร้าง socket address โดยจะคืนกลับค่า socket address ของ client ที่ร้องขอสร้างการเชื่อมต่อเข้ามา

len เป็น pointer ไปยัง integer โดยจะคืนกลับค่าความยาวของตัวแปรโครงสร้าง *cliaddr*

3.7 การรับ/ส่งข้อมูลในโปรโตคอล TCP โดยใช้ฟังก์ชัน send() และ recv()

หลังจากที่มีการสร้างการเชื่อมต่อระหว่าง client และ server แล้ว จะเห็นว่าทั้ง server และ client จะมี socket ของตัวเองที่เชื่อมต่ออยู่กับ socket ของอีกฝ่ายหนึ่งอยู่แล้ว ดังนั้นในการรับและส่งข้อมูลระหว่างกันจึงสามารถทำได้โดยอาศัยการรับ/ส่งข้อมูลผ่านทาง socket ที่ทำการเชื่อมต่อไว้แล้ว

ฟังก์ชัน send()

ฟังก์ชัน send() ใช้ในการส่งข้อมูลผ่านทาง socket ที่ได้ทำการเชื่อมต่อ ไปยังอีกฝ่ายหนึ่งของการเชื่อมต่อนั้น โดยจะมีรูปแบบดังนี้

```
#include <sys/socket.h>
int send(int sockfd, const char *buf, int len, int flags);
```

sockfd เป็น socket handler สำหรับ socket ที่ต้องการส่งข้อมูลไปยังอีกฝ่ายหนึ่งของการเชื่อมต่อ

buf เป็นข้อมูลที่ต้องการส่งไปยังผู้รับ

len เป็นค่าความยาวของข้อมูล *buf*

flags เป็นค่าคุณสมบัติเพิ่มเติมในการส่งข้อมูล

โดยฟังก์ชัน send() จะคืนกลับค่าความยาวของข้อมูลที่ได้ส่ง เป็นจำนวน byte ซึ่งอาจจะน้อยกว่าค่าความยาวของข้อมูลที่ต้องการส่ง (*len*) ก็ได้ หรือค่าแสดงความผิดพลาดอื่นๆ

ฟังก์ชัน recv()

ฟังก์ชัน recv() ใช้ในการรับข้อมูลที่ส่งมาทาง socket ที่ได้ทำการเชื่อมต่อ โดยจะมีรูปแบบดังนี้

```
#include <sys/socket.h>
int recv(int sockfd, char *buf, int len, int flags);
```

sockfd เป็น socket handler สำหรับ socket ที่ต้องการรับข้อมูลที่ส่งมาจากอีกฝ่ายหนึ่งของการเชื่อมต่อ

buf เป็นข้อมูลที่ได้รับจากผู้ส่ง

len เป็นค่าความยาวของข้อมูลที่สามารถรับได้สูงสุด

flags เป็นค่าคุณสมบัติเพิ่มเติมในการรับข้อมูล

โดยฟังก์ชัน recv() จะคืนกลับค่าความยาวของข้อมูลที่ได้รับจริง เป็นจำนวน byte หรือค่าแสดงสถานะความผิดพลาดอื่นๆ

3.8 การยกเลิกการเชื่อมต่อในโปรโตคอล TCP โดยใช้ฟังก์ชัน close()

หลังจากการติดต่อสื่อสารโดยการสร้างการเชื่อมต่อเสร็จสิ้นแล้ว ทั้ง server และ client สามารถที่จะปิดการเชื่อมต่อที่สร้างขึ้นได้อย่างเป็นอิสระต่อกัน นั่นคือหากฝ่ายใดฝ่ายหนึ่งไม่มีการส่งข้อมูลไปยังอีกฝ่ายหนึ่งอีกต่อไปก็จะสามารถปิดการเชื่อมต่อที่สร้างไว้ทางฝ่ายของตน แต่จะสามารถรับข้อมูลที่ส่งเข้ามาได้จนกว่าอีกฝ่ายหนึ่งจะปิดการเชื่อมต่อ

รูปแบบของฟังก์ชัน close() เป็นดังนี้

```
#include <unistd.h>
int close(int sockfd);
```

sockfd เป็น socket handler สำหรับ socket ที่ต้องการปิดการเชื่อมต่อที่ได้เคยสร้างไว้ที่ socket นี้

ทั้งนี้ ฟังก์ชัน close() จะคืนกลับค่า 0 หากไม่เกิดข้อผิดพลาด หรือค่าแสดงความผิดพลาดอื่นๆหากเกิดข้อผิดพลาดในการปิดการเชื่อมต่อ

3.9 การรับ/ส่งข้อมูลในโปรโตคอล UDP โดยใช้ฟังก์ชัน sendto() และ recvfrom()

ในการติดต่อสื่อสารโดยใช้โปรโตคอล UDP จะไม่มีการสร้างการเชื่อมต่อระหว่างผู้ส่งและผู้รับ ดังนั้นในการส่งข้อมูลไปยังผู้รับสามารถทำได้โดยการระบุ address อ้างอิงของผู้รับ และส่งข้อมูลไปยัง address อ้างอิงนั้น ซึ่งจะสามารถส่ง/รับข้อมูลผ่านทาง socket ที่สร้างขึ้นไปยังผู้รับคนใดก็ได้

ฟังก์ชัน sendto()

ฟังก์ชัน sendto() ใช้ในการส่งข้อมูลไปยังผู้รับผ่านทาง socket ที่ได้สร้างไว้ โดยใช้โปรโตคอล UDP มีรูปแบบดังนี้

```
#include <sys/socket.h>
ssize_t sendto(int sockfd, cont char *buf, size_t nbytes, int flags,
        const struct sockaddr *to, socklen_t addrlen);
```

sockfd เป็น socket handler สำหรับ socket ที่สร้างขึ้นที่ต้องการใช้ในการส่งข้อมูลออกไปยังผู้รับ

buf เป็นข้อมูลที่ต้องการส่งไปยังผู้รับ

nbytes เป็นความยาวของข้อมูล *buf*

flags เป็นค่าคุณสมบัติเพิ่มเติมอื่นๆในการส่งข้อมูล

to เป็นตัวแปรโครงสร้าง socket address ที่ระบุถึง address อ้างอิง เพื่อใช้ในการอ้างอิงไปยังผู้รับได้ถูกต้อง

addrlen เป็นค่าความยาวของตัวแปรโครงสร้าง to

ฟังก์ชัน sendto() จะคืนกลับค่าความยาวของข้อมูลที่ส่งไปได้จริง หน่วยเป็น byte หากสามารถส่งข้อมูลได้สำเร็จ หรือค่าความผิดพลาดอื่นๆหากเกิดความผิดพลาด

ฟังก์ชัน recvfrom()

ฟังก์ชัน recvfrom() ใช้ในการรับข้อมูลที่ส่งเข้ามาผ่านทาง socket ที่ได้สร้างไว้ โดยใช้โปรโตคอล UDP มีรูปแบบดังนี้

```
#include <sys/socket.h>
ssize_t recvfrom(int sockfd, void *buf, size_t nbytes, int flags,
        struct sockaddr *from, socklen_t addrlen);
```

sockfd เป็น socket handler สำหรับ socket ที่สร้างขึ้นที่ต้องการใช้ในการรับข้อมูล

buf เป็นข้อมูลที่ได้รับ

nbytes เป็นความยาวของข้อมูลที่สามารถรับได้สูงสุด

flags เป็นค่าคุณสมบัติเพิ่มเติมอื่นๆในการรับข้อมูล

from เป็นตัวแปรโครงสร้าง socket address ที่คืนกลับค่า address อ้างอิงของผู้ส่งข้อมูลนี้

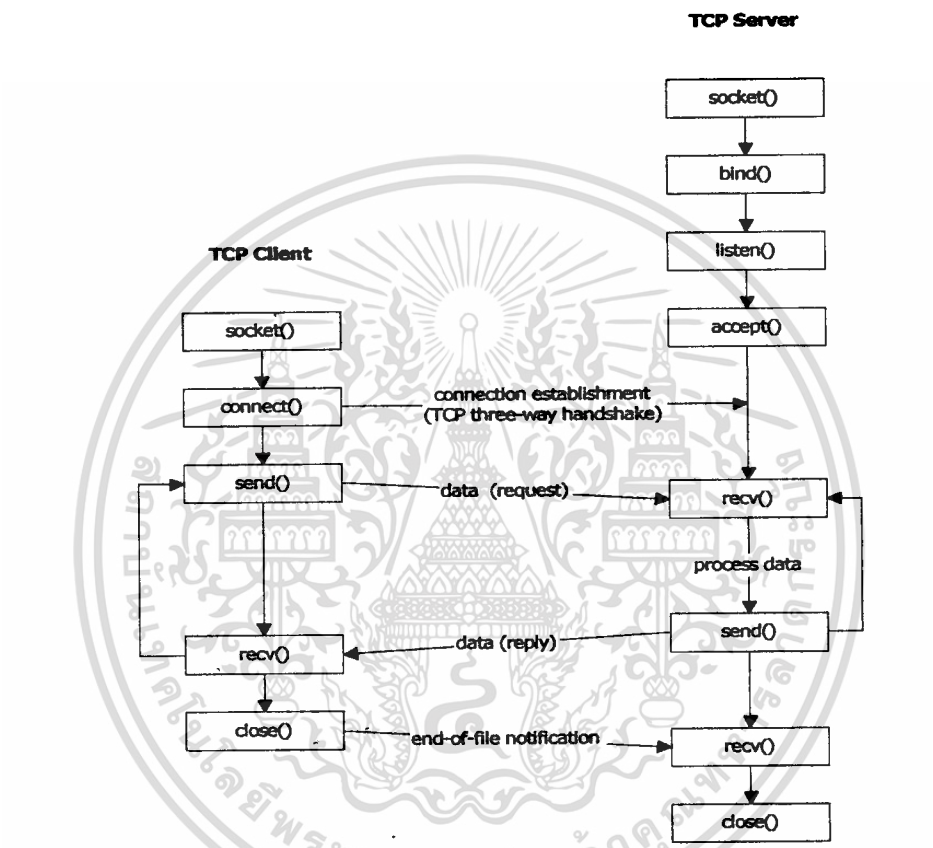
addrlen เป็นค่าความยาวของตัวแปรโครงสร้าง from

ฟังก์ชัน recvfrom() จะคืนกลับค่าความยาวของข้อมูลที่ได้รับจริง หน่วยเป็น byte หากสามารถรับข้อมูลได้สำเร็จ หรือค่าความผิดพลาดอื่นๆหากเกิดความผิดพลาด

3.10 ตัวอย่างโปรแกรม TCP echo server, echo client

โปรแกรม echo server เป็นโปรแกรมตัวอย่างที่ง่ายต่อการทำความเข้าใจ โดยการทำงานจะมีขั้นตอนดังต่อไปนี้

1. client รับค่าข้อความจากผู้ใช้ทางคีย์บอร์ด และส่งข้อความนั้นไปให้กับ server
2. เมื่อ server ได้รับข้อความจาก client แล้ว ก็จะส่งข้อความนั้นกลับคืนไปให้กับ client
3. client รับข้อความที่คืนกลับมาจาก server แล้วจึงนำมาแสดงผลออกทางจอภาพ



รูปที่ 3.5 การทำงานของ TCP echo server และ TCP echo client

โปรแกรม TCP echo Server : ฟังก์ชัน main()

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/socket.h>
5 #include <sys/socket.h>
6 #include <arpa/inet.h>
7 #include <netinet/in.h>
```

```
 8 int main(int argc, char **argv)
 9 {
10  int   listenfd, connfd;
11  int   clilen;
12  struct sockaddr_in servaddr, cliaddr;

13  listenfd = socket(AF_INET, SOCK_STREAM, 0);
14  bzero(&servaddr, sizeof(servaddr));
15  servaddr.sin_family = AF_INET;
16  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
17  servaddr.sin_port = htons(8888);

18  if (bind(listenfd, (struct sockaddr*) &servaddr,
              sizeof(servaddr)) != 0 ) {
19        perror("bind error");
20        exit(1);
21  }
22  if (listen(listenfd, LISTENQ) != 0) {
23        perror("lieten error");
24        exit(1);
25  }
26  for(;;) {
27        clilen = sizeof(cliaddr);
28        connfd = accept(listenfd, (struct sockaddr*)
                        &cliaddr, &clilen);
29        str_echo(connfd);
30  }
31 }
```

รูปที่ 3.6 TCP echo server: main() function

1. echo server สร้าง socket ชนิด stream socket โดยฟังก์ชัน socket() และกำหนด socket handler สำหรับอ้างถึง socket นั้นๆ (บรรทัดที่ 12)
2. กำหนด address สำหรับใช้อ้างอิงในการติดต่อให้กับ socket (หมายเลข port, IP address) โดยฟังก์ชัน `bind()` (บรรทัดที่ 14 - 21) โดยกำหนดให้ใช้ address family เป็น AF_INET และสามารถรับการเชื่อมต่อได้ทางหมายเลข port 8888
3. จากนั้นทำการกำหนดให้ socket `listenfd` พร้อมที่จะรับการติดต่อจาก client โดยฟังก์ชัน `listen()` และรอจนกว่าจะมีการติดต่อมาจาก client (บรรทัดที่ 22 - 25)
4. เมื่อมีการขอเชื่อมต่อ จาก client แล้ว นั่นคือหลังจากขบวนการ TCP Three-way handshake แล้ว จะมีการส่งสัญญาณจาก kernel ของระบบปฏิบัติการ ไปยัง server process ผ่านทาง socket ที่สร้างไว้ (`listenfd`) echo server จะเรียกใช้ฟังก์ชัน `accept()` เพื่อทำการเชื่อมต่อกับ echo client process นั้น และสร้าง socket ใหม่ (`connfd`) ที่เชื่อมต่อแล้วระหว่าง echo server และ echo client process ดังกล่าว (บรรทัดที่ 28)

โปรแกรม TCP echo Server : ฟังก์ชัน str_echo()

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/socket.h>
5 #include <sys/socket.h>
6 #include <arpa/inet.h>
7 #include <netinet/in.h>

8 #define MAXLINE 4096
9 #define LISTENQ 1024

10 void str_echo(int sockfd)
11 {
12  size_t       n;
13  char         line[MAXLINE];

14   for(;;) {
15        memset(line,0,MAXLINE);
16        if ((n = recv(sockfd, line, MAXLINE, 0)) == 0) {
17             return;
18        }
19        send(sockfd, line, n, 0);
20   }
21 }
```

รูปที่ 3.7 TCP echo server: ฟังก์ชัน str_echo()

ฟังก์ชัน str_echo() เป็นฟังก์ชันที่ทำหน้าที่ในการประมวลผลข้อมูลที่ส่งมาจาก client นั่นคือการส่งค่าข้อมูลนั้นกลับคืนไปให้กับ client

1. echo server รับข้อมูลที่จะส่งเข้ามา echo client process ได้ ผ่านทาง socket ที่สร้างการเชื่อมต่อแล้ว โดยฟังก์ชัน `recv()` (บรรทัดที่ 16)
2. echo server ไม่ทำการประมวลผลใดๆกับข้อมูลที่ได้รับ แต่จะส่งข้อมูลนั้นกลับไปยัง echo client process ทันที โดยใช้ฟังก์ชัน `send()` โดยไม่ต้องระบุ IP address และหมายเลข port ของ echo client process เนื่องจาก socket ได้ถูกเชื่อมต่อระหว่าง echo server process และ echo client process อยู่แล้ว (บรรทัดที่ 19)
3. หลังจากนั้น echo server จะกลับไปรอรับข้อมูลอื่นๆที่จะส่งเข้ามา และทำงานในขั้นตอนที่ 1 – 3 ต่อไป

โปรแกรม TCP echo Client : ฟังก์ชัน main()

```
 1 #include <errno.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <string.h>
 5 #include <sys/types.h>
 6 #include <sys/socket.h>
 7 #include <arpa/inet.h>
 8 #include <netinet/in.h>

 9 int main(int argc, char **argv)
10 {
11  int   sockfd, n, i;
12  struct sockaddr_in servaddr;

13  if (argc != 2) {
14       printf("usage: a.out <IPADDRESS>\n");

15       exit(1);
16  }
17  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) <0 ) {
18       perror("socket eror");
19       exit(1);
20  }

21  bzero(&servaddr, sizeof(struct sockaddr));
22  servaddr.sin_family = AF_INET;
23  servaddr.sin_port = htons(8888);

24  if (inet_aton(argv[1], &(servaddr.sin_addr))==0) {
25       printf("inet_aton error\n");
26       exit(1);
27  }
28  i = connect(sockfd, (struct sockaddr*) &servaddr,
         sizeof(servaddr));
29  if (i <0) {
30       perror("connect eror");
31       exit(1);
32  }

33  str_cli(stdin, sockfd);

34  exit(0);
35 }
```

รูปที่ 3.8 TCP echo client : ฟังก์ชัน main()

โปรแกรม TCP echo client จะทำการเชื่อมต่อไปยัง TCP server โดยในการใช้งานจะต้องระบุ IP address ของ server ที่ต้องการติดต่อด้วย เช่น "C:\ tcp_client 161.200.152.24" ในกรณีที่

โปรแกรม TCP echo client (ชื่อ tcp_client.exe) ต้องการติดต่อไปยัง echo server ที่เปิดรอรับการเชื่อมต่ออยู่ที่เครื่องคอมพิวเตอร์ 161.200.152.24 เป็นต้น

1. client จะสร้าง socket (sockfd) เพื่อใช้ในการติดต่อ เช่นเดียวกับ server process (บรรทัดที่ 17)
2. ทำการขอสร้างการเชื่อมต่อ (connection) กับ echo server โดยฟังก์ชัน connect() ระบุถึงปลายทาง (server process) โดยใช้ address อ้างอิงที่ server กำหนด (IP address และ หมายเลข port ของ server process) โดยถูกระบุไว้เป็น argument ในขณะเรียกใช้โปรแกรม (บรรทัดที่ 21 - 28)
3. เมื่อ echo server ยอมรับการขอเชื่อมต่อ (โดยฟังก์ชัน accept()) แล้ว echo client จะสามารถรับ/ส่งข้อมูลระหว่าง client กับ server ได้ต่อไป โดยผ่านทาง socket ที่ได้สร้างขึ้น

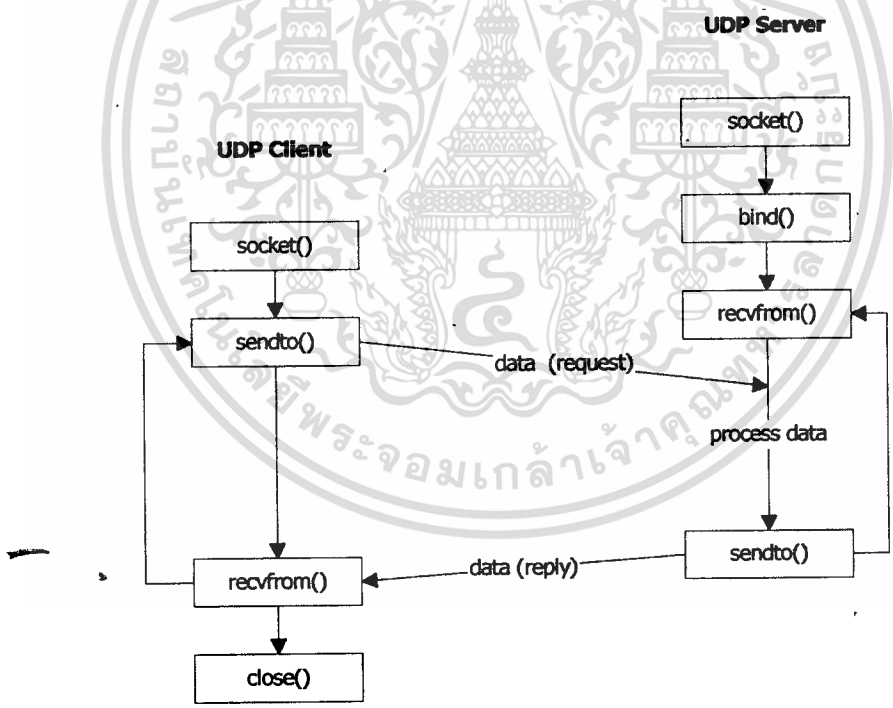
โปรแกรม TCP echo Client : ฟังก์ชัน str_cli()

```
 1 #include <errno.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <string.h>
 5 #include <sys/types.h>
 6 #include <sys/socket.h>
 7 #include <arpa/inet.h>
 8 #include <netinet/in.h

 9 #define MAXLINE 4096

10 void str_cli(FILE *fp, int sockfd)
11 {
12  char  sendline[MAXLINE], recvline[MAXLINE];

13  while (fgets(sendline, MAXLINE, fp) != NULL) {
14        send(sockfd, sendline, strlen(sendline), 0);
15        if (recv(sockfd, recvline, MAXLINE, 0) == 0) {
16              printf("str_cli: server terminated
                      prematurely\n");
17              break;
18        }
19        fputs(recvline, stdout);
20        memset(sendline,0,MAXLINE);
21        memset(recvline,0,MAXLINE);
22  }
23 }
```

รูปที่ 3.9 TCP echo client : ฟังก์ชัน str_cli()

1. echo client รับข้อมูลจากผู้ใช้ทางคีย์บอร์ด (บรรทัดที่ 13) และส่งข้อความนั้นไปยัง echo server ผ่านทาง socket ที่สร้างการเชื่อมต่อแล้ว โดยฟังก์ชัน `send()` (บรรทัดที่ 14)
2. หลังจากนั้น echo client จะรอรับข้อมูลที่ส่งกลับมาจาก echo server โดยฟังก์ชัน `recv()` (บรรทัดที่ 15)
3. เมื่อ echo client ได้รับข้อมูลที่ส่งกลับมาจาก echo server แล้ว จึงนำข้อมลนั้นไปแสดงผลยังจอภาพ และจะกลับไปรอรับข้อมูลอื่นๆจากผู้ใช้ และส่งข้อมูลนั้นไปให้กับ server และทำงานในขั้นตอนที่ 1 – 3 ต่อไป

3.11 ตัวอย่างโปรแกรม UDP echo server, echo client



รูปที่ 3.10 การทำงานของ UDP server และ UDP client

โปรแกรม UDP echo server และ client ทำงานในลักษณะเช่นเดียวกับ TCP echo server และ client เพียงแต่ UDP echo จะทำการติดต่อสื่อสารโดยใช้โปรโตคอล UDP ผ่านทาง socket ชนิด datagram socket แทน stream socket

โปรแกรม UDP echo Server : ฟังก์ชัน main()

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <sys/types.h>
 4 #include <sys/socket.h>
 5 #include <sys/socket.h>
 6 #include <arpa/inet.h>
 7 #include <netinet/in.h>

 8 int main(int argc, char **argv)
 9 {
10  int   sockfd;
11  struct sockaddr_in servaddr, cliaddr;

12  sockfd = socket(AF_INET, SOCK_DGRAM, 0);

13  bzero(&servaddr, sizeof(servaddr));
14  servaddr.sin_family = AF_INET;
15  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
16  servaddr.sin_port = htons(8888);

17  if (bind(sockfd, (struct sockaddr*) &servaddr,
            sizeof(servaddr)) != 0 ) {
18      perror("bind error");
19      exit(1);
20  }
21  dg_echo(sockfd, (struct sockaddr *) &cliaddr,
        sizeof(cliaddr));

22 }
```

รูปที่ 3.11 UDP echo server : ฟังก์ชัน main()

1. โปรแกรม UDP echo server ทำการสร้าง socket โดยฟังก์ชัน `socket()` เป็นชนิด datagram socket (บรรทัดที่ 12)
2. กำหนด address สำหรับใช้อ้างอิงในการติดต่อให้กับ socket (หมายเลข port, IP address) โดยฟังก์ชัน bind() (บรรทัดที่ 14 – 20) โดยกำหนดให้ใช้ address family เป็น AF_INET และสามารถรับการเชื่อมต่อได้ทางหมายเลข port 8888
3. UDP server สามารถที่จะรับข้อมูลที่ส่งเข้ามาจาก client ใดๆก็ได้โดยฟังก์ชัน `redvfrom()` และสามารถส่งข้อมูลออกไปยัง client ใดๆก็ได้ทาง socket ที่สร้างขึ้นโดยฟังก์ชัน `sendto()`

จะเห็นว่า การติดต่อสื่อสารโดยใช้โปรโตคอล UDP (datagram socket) นั้น ไม่ต้องมีการสร้างการเชื่อมต่อก่อนการรับ/ส่งข้อมูลแต่อย่างใด ทั้งนี้ทำให้สามารถส่งข้อมูลไปยังผู้รับใดๆก็ได้ผ่านทาง socket ที่สร้างขึ้นเพียง socket เดียว โดยจะต้องมีการระบุ address อ้างอิงของผู้รับ สำหรับแต่ละข้อความที่ส่งออกไป

โปรแกรม UDP echo Server : ฟังก์ชัน dg_echo()

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <sys/types.h>
 4 #include <sys/socket.h>
 5 #include <sys/socket.h>
 6 #include <arpa/inet.h>
 7 #include <netinet/in.h>

 8 #define MAXLINE 4096
 9 #define LISTENQ 1024

10 void dg_echo(int sockfd, struct sockaddr *pcliaddr,
   int cliaddr)
11 {
12  int    n, len;
13  char   mesg[MAXLINE];

14  for(;;) {
15        memset(mesg,0,MAXLINE);
16        if ((n = recvfrom(sockfd, mesg, MAXLINE, 0,
                    pcliaddr, &cliaddr)) <0) {
17              perror("recvfrom error:");
18              return;
19        }
20        if (sendto(sockfd, mesg, MAXLINE, 0, pcliaddr,
                    cliaddr) <0 ) {
21              perror("sendto error:");
22              return;
23        }
24  }
25 }
```

รูปที่ 3.12 UDP echo server : ฟังก์ชัน dg_echo()

1. UDP echo server รับข้อมูลที่จะส่งเข้ามา echo client ผ่านทาง socket ที่สร้างไว้ โดยฟังก์ชัน `recvfrom()` (บรรทัดที่ 16) และจะได้รับค่า address อ้างอิงของ client จากการคืนค่าของฟังก์ชัน `recvfrom()`

2. echo server ไม่ทำการประมวลผลใดๆกับข้อมูลที่ได้รับ แต่จะส่งข้อมูลนั้นกลับไปยัง echo client process ทันที โดยใช้ฟังก์ชัน `sendto()` โดยระบุ address อ้างอิงของ client (IP address และหมายเลข port) จาก address ที่ได้รับจากฟังก์ชัน `recvfrom()` (บรรทัดที่ 20)
3. หลังจากนั้น echo server จะกลับไปรอรับข้อมูลอื่นๆที่จะส่งเข้ามา (จาก client เดียวกันนี้หรือ client อื่นๆ) และทำงานในขั้นตอนที่ 1 – 3 ต่อไป

โปรแกรม UDP echo Client : ฟังก์ชัน main()

โปรแกรม UDP client ทำการสร้าง socket เพื่อใช้ในการติดต่อสื่อสาร และส่งข้อมูล ไปยัง UDP server โดยระบุ IP address และหมายเลข port ของ server ที่ต้องการติดต่อด้วย จากนั้นจะรอรับข้อมูลที่ส่งกลับมาจาก server หลังจากที่ server ได้ประมวลผลแล้ว

```
 1 #include <errno.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <string.h>
 5 #include <sys/types.h>
 6 #include <sys/socket.h>
 7 #include <arpa/inet.h>
 8 #include <netinet/in.h>

 9 int main(int argc, char **argv)
10 {
11  int   sockfd, n, i;
12  struct sockaddr_in servaddr;
13  if (argc != 2) {
14        printf("usage: a.out <IPADDRESS>\n");
15        exit(1);
16  }
17  if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) <0 ) {
18        perror("socket eror");
19        exit(1);
20  }
21  bzero(&servaddr, sizeof(struct sockaddr));
22  servaddr.sin_family = AF_INET;
23  servaddr.sin_port = htons(8888);
24  if (inet_aton(argv[1], &(servaddr.sin_addr))==0) {
25        printf("inet_aton error\n");
26        exit(1);
27  }
28  dg_cli(stdin, sockfd, (struct sockaddr*) &servaddr,
        sizeof(servaddr));
29  exit(0);
30 }
```

รูปที่ 3.13 UDP echo client : ฟังก์ชัน main()

1 UDP echo client จะสร้าง socket (sockfd) เพื่อใช้ในการติดต่อ เช่นเดียวกับ server process (บรรทัดที่ 17)

2 ทำการเตรียมตัวแปรโครงสร้าง socket address (servaddr) สำหรับเป็น address อ้างอิงของ server เพื่อใช้ในการส่งข้อมูลไปยัง server ตาม address อ้างอิงที่ระบุ

3 UDP echo client สามารถรับ/ส่งข้อมูลไปยัง server ได้ทันทีผ่านทาง socket ที่สร้างขึ้น

โปรแกรม UDP echo Server : ฟังก์ชัน dg_echo()

```
 1 #include <errno.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <string.h>
 5 #include <sys/types.h>
 6 #include <sys/socket.h>
 7 #include <arpa/inet.h>
 8 #include <netinet/in.h>

 9 #define MAXLINE 4096

10 void dg_cli(FILE* fp, int sockfd, struct sockaddr* pservaddr,
     int servlen)
11 {
12   int    n;
13   char   sendline[MAXLINE], recvline[MAXLINE+1];

14         while (fgets(sendline, MAXLINE, fp) != NULL) {
15                if (sendto(sockfd, sendline, strlen(sendline), 0,
                                 pservaddr, servlen) <0 ){
16                      perror("sendto error:");
17                      exit(1);
18                }
19                if ((n = recvfrom(sockfd, recvline, MAXLINE, 0,
                                 pservaddr, &servlen)) <0) {
20                      perror("recvfrom error:");
21                      exit(1);
22                }
23                recvline[n] = 0;
24                fputs(recvline, stdout);
25                memset(sendline,0,MAXLINE);
26                memset(recvline,0,MAXLINE);
27         }
28 }
```

รูปที่ 3.14 UDP echo client : ฟังก์ชัน dg_echo()

1. UDP echo client รับข้อมูลจากผู้ใช้ทางคีย์บอร์ด (บรรทัดที่ 14) และส่งข้อความนั้นไปยัง echo server ผ่านทาง socket โดยระบุ address อ้างอิงของ server (IP address และหมายเลข port) โดยฟังก์ชัน `sendto()` (บรรทัดที่ 15)
2. หลังจากนั้น echo client จะรอรับข้อมูลที่ส่งกลับมาจาก echo server โดยฟังก์ชัน `recvfrom()` (บรรทัดที่ 19)
4. เมื่อ echo client ได้รับข้อมูลที่ส่งกลับมาจาก echo server แล้ว จึงนำข้อมลนั้นไปแสดงผลยังจอภาพ และจะกลับไปรอรับข้อมูลอื่นๆจากผู้ใช้ และส่งข้อมูลนั้นไปให้กับ server และทำงานในขั้นตอนที่ 1 – 3 ต่อไป

ในการเขียนโปรแกรมสำหรับทำงานและสื่อสารกันผ่านทางระบบเครือข่าย TCP/IP นั้น โปรแกรมจะทำการรับ/ส่งข้อมูลได้ในหลายลักษณะขึ้นกับชนิดของ socket ที่ใช้ ที่นิยมคือ Stream socket ใช้ protocol TCP ในการติดต่อสื่อสาร ซึ่งมีความถูกต้องสูง และมีการสร้างการเชื่อมต่อระหว่างต้นทางและปลายทางในการติดต่อ และ Datagram socket ใช้ protocol UDP ในการติดต่อสื่อสาร โดยจะไม่มีการตรวจสอบความถูกต้อง และไม่มีการสร้างการเชื่อมต่อระหว่างต้นทางและปลายทาง ขั้นตอนในการทำงานของโปรแกรมนั้น โปรมแกรมจะต้องสร้างช่องทางในการสื่อสาร (socket) และสร้างการเชื่อมต่อระหว่างต้นทางและปลายทาง (ในกรณีที่เป็น Stream socket) จากนั้น โปรแกรมจึงสามารถส่ง/รับข้อมูลผ่านทาง socket ที่สร้างไว้กับโปรแกรมที่ทำงานอยู่อีกฝั่งหนึ่งได้

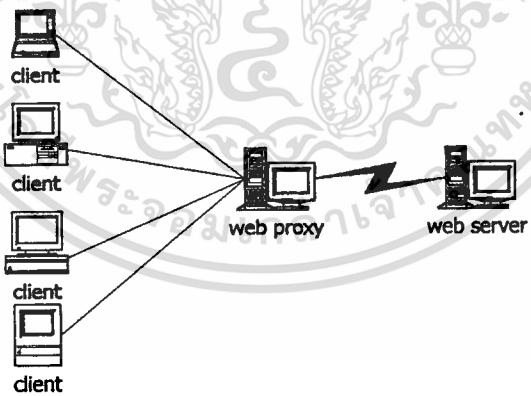
ทั้งนี้รูปแบบและรายละเอียดข้อบังคับต่างๆ เกี่ยวกับการใช้งาน socket นั้น อาจมีความแตกต่างกันบ้างตามภาษาและเครื่องมือที่ใช้ในการพัฒนาโปรแกรมที่เลือกใช้ ซึ่งจะต้องมีการศึกษารายละเอียดเฉพาะของแต่ละ API ที่ใช้ แต่จะมีลักษณะการทำงานในรูปแบบเดียวกันกับ Berkeley socket ดังที่ได้กล่าวไปแล้ว

บทที่ 4

WWW proxy

จากการบริการ World-Wide Web ได้รับความนิยมอย่างสูง ทำให้มีผู้ใช้เป็นจำนวนมาก จึงส่งผลให้ประสิทธิภาพของระบบเครือข่าย และของ WWW server ที่ต้องให้บริการอยู่ตลอดเวลาลดน้อยลง เนื่องจากต้องให้บริการผู้ใช้ที่มีจำนวนมากขึ้น อีกทั้งข้อมูลต่างๆก็มีขนาดใหญ่ขึ้นและมีความซับซ้อนมากขึ้นด้วย จึงมีการคิดพัฒนาเทคนิคต่างๆเพื่อช่วยเพิ่มประสิทะภาพในการใช้งาน Internet

Proxy เป็นเทคนิควิธีหนึ่งในการให้บริการการเข้าถึงเครือข่าย Internet ให้แก่เครื่องคอมพิวเตอร์ 1 เครื่องหรือมากกว่าซึ่งไม่ได้ต่อเชื่อมกับเครือข่าย Internet โดยตรง แต่มีเพียงการเชื่อมต่อกับเครื่องคอมพิวเตอร์ที่ทำหน้าที่เป็น proxy ซึ่งเชื่อมต่อกับเครือข่าย Internet หรือ proxy อื่นเท่านั้น



รูปที่ 4.1 แสดงลักษณะการเชื่อมต่อของ client ผ่านทาง proxy

โดยสามารถแบ่งตามลักษณะการทำงานและประโยชน์ได้เป็น 2 ชนิดคือ

- ความสามารถในการให้บริการเชื่อมต่อกับเครือข่าย Internet ให้แก่เครื่องคอมพิวเตอร์โดยทางอ้อม
- ความสามารถในการเพิ่มประสิทธิภาพในการเรียกใช้ข้อมูลจากเครือข่าย Internet โดยอาศัย cache ของข้อมูลที่เก็บไว้ที่เครื่อง proxy server

4.1 Proxy ทำหน้าที่เป็น Gateway

proxy gateway เป็นโปรแกรมที่ทำงานอยู่บนเครื่องคอมพิวเตอร์แบบ dual-homed host หรือ bastion host ซึ่งก็คือ เครื่องคอมพิวเตอร์ที่สามารถติดต่อสื่อสารกับเครื่องอื่นๆภายในเครือข่ายได้ และสามารถติดต่อสื่อสารกับเครือข่ายภายนอก (เช่น Internet) ได้ด้วย (มี network interface มากกว่า 1 interface) ทำให้ proxy server สามารถทำหน้าที่เป็นตัวกลางในการเชื่อมโยงเครื่องคอมพิวเตอร์ที่อยู่ในเครือข่ายภายใน (เช่น LAN, Intranet ฯลฯ) กับเครือข่ายภายนอก (เช่น Internet) ได้

โดยในการทำงานนั้น เมื่อเครื่องคอมพิวเตอร์แต่ละเครื่อง (client) ที่อยู่ในเครือข่ายภายในมีความต้องการติดต่อสื่อสารกับเครื่องคอมพิวเตอร์ผู้ให้บริการที่อยู่ในเครือข่าย Internet ก็จะทำการสื่อสารผ่านทาง proxy server แทน โดย proxy server จะประมวลผลการร้องขอ (request) ที่ได้รับจาก client และพิจารณาว่าจะอนุญาติให้มีการติดต่อสื่อสารกันได้หรือไม่ หากอนุญาติ ก็จะทำการติดต่อไปยังผู้ให้บริการปลายทาง (server) แทน client นั้น จากนั้นเครื่องคอมพิวเตอร์ผู้ให้บริการปลายทางจะทำการประมวลผลการร้องขอนั้น และตอบกลับการร้องขอ (response) กลับไปยัง proxy server จากนั้น proxy server ก็จะนำข้อมูลเหล่านั้นส่งกลับไปให้กับ client ที่ร้องขอต่อไป

ซึ่งจะเห็นว่าในการติดต่อสื่อสารกันนั้น หากมองในมุมมองของ client แล้ว client สามารถติดต่อกับ proxy server เสมือนการติดต่อสื่อสารกับเครื่องคอมพิวเตอร์ที่ให้บริการที่ร้องขอนั้นจริงๆ และหากมองในมุมมองของ server ผู้ให้บริการนั้น server ผู้ให้บริการจะทำการติดต่อสื่อสารและให้บริการกับ proxy server เสมือนการสื่อสารกับ client ตามปกติ โดยที่ server ผู้ให้บริการไม่ต้องทราบว่า client ที่ขอใช้บริการที่แท้จริงคือใคร อยู่ที่ใดเลย

4.2 Web Caching

WWW proxy เป็น proxy server ที่มุ่งเน้นให้บริการเฉพาะบริการ World Wide Web เท่านั้น โดยที่เครื่องคอมพิวเตอร์ที่ต้องการใช้บริการ World Wide Web อาศัยการเชื่อมต่อของ WWW proxy จะทำการตั้งค่า (configure) เพื่อให้ส่งการร้องขอบริการ World Wide Web (HTTP request) ไปยัง WWW proxy แทนการส่งไปยัง WWW server ที่ให้บริการโดยตรง

โดยจากลักษณะการทำงานของ proxy server ที่เป็นตัวกลางในการติดต่อสื่อสารระหว่าง client ผู้ร้องขอใช้บริการและ server ผู้ให้บริการนั้น ทำให้ WWW proxy สามารถนำวิธีการ caching มาใช้เพื่อเป็นการเพิ่มประสิทธิภาพในการใช้งานทรัพยากรต่างๆให้มากขึ้น อาศัยสมมติฐานว่าการร้องขอทรัพยากรนั้น จะมีการร้องขอทรัพยากรที่ซ้ำๆกันอยู่บ่อยครั้ง ดังนั้น WWW proxy จะทำการเก็บข้อมูลทรัพยากรต่างๆที่เคยมีการรับ/ส่งกันระหว่าง web client (เช่น WWW browser) และ

WWW server ที่ส่งผ่านทาง proxy ไว้ใน cache ของ proxy เอง และเมื่อมีการร้องขอทรัพยากรเดียวกันนี้จาก web client ในภายหลัง WWW proxy จะทำการให้บริการต่อการร้องขอของ client นั้นเองด้วยข้อมูลที่เก็บอยู่ใน cache แทนการส่งการร้องขอนั้น ไปยัง WWW server อีกครั้งหนึ่ง

จะเห็นว่าการใช้งาน proxy caching สามารถเพิ่มความเร็วในการเรียกใช้ทรัพยากร และลดการเข้าถึง server ผู้ให้บริการได้ในระดับหนึ่ง โดยสามารถแยกข้อดีได้ดังนี้

1. ลดเวลาในการเข้าถึงข้อมูลให้แก่ client เนื่องจาก proxy server ต่อเชื่อมกับ client ในระยะทางที่ใกล้กว่าระยะทางเชื่อมต่อระหว่าง client กับเครื่อง server ที่ให้บริการทรัพยากรนั้นๆ จริงๆ

2. ลดการใช้งานของเครือข่ายลง เนื่องจากการให้บริการของ proxy ไม่ต้องอาศัยการติดต่อกับเครือข่ายภายนอก และยังใช้ทรัพยากรเครือข่ายน้อยกว่าการติดต่อเพื่อขอใช้บริการ และการให้บริการจาก server ผู้ให้บริการจริงๆ

3. ลดความต้องการในการเข้าถึง server ผู้ให้บริการลง เนื่องจาก proxy caching สามารถให้บริการแก่การร้องขอ (request) ของ client ส่วนหนึ่งโดยอาศัยข้อมูลจาก cache ได้ ทำให้ลดการเข้าถึง WWW server ที่ให้บริการจริงลง

ลักษณะของ web caching

สามารถแยกออกได้เป็น 2 ลักษณะใหญ่ๆ ซึ่งมีลักษณะการใช้งานและประโยชน์ที่แตกต่างกัน ดังนี้

1) server cache อาศัยการทำงานของเครื่องคอมพิวเตอร์ ที่ให้บริการเป็นตัวกลางรับ/ส่งข้อมูลแก่เครื่อง client จำนวนมาก และยังสามารถทำหน้าที่เป็น gateway สำหรับเชื่อมต่อเครื่อง client เหล่านั้นกับระบบเครือข่ายภายนอกได้ด้วย ซึ่งจะมีการเรียกใช้และติดต่อสื่อสารกันระหว่าง client จำนวนมาก กับเครื่อง WWW server ที่ให้บริการซึ่งอยู่ในเครือข่ายภายนอก (เช่น ใน Internet) ทั้งนี้จะเห็นว่าการ caching ข้อมูลแบบ server cache นั้น จะเก็บข้อมูลที่มีการรับ/ส่งกันระหว่าง client กับ WWW server เป็นจำนวนมาก จึงมีโอกาสที่จะมีการร้องขอข้อมูลซ้ำๆจาก client หลายๆ เครื่องได้มาก ทำให้ server cache สามารถช่วยเพิ่มความเร็วในการเข้าถึงข้อมูล World-Wide Web และช่วยลดความคับคั่งของเครือข่ายได้มากพอสมควร

2) client cache อาศัยการเก็บข้อมูลที่เคยมีการรับ/ส่งกันไว้ที่เครื่อง client เองโดย user agent (เช่น โปรแกรม WWW browser) เป็นผู้จัดการเอง หรืออาจอาศัย โปรแกรม WWW proxy ที่เป็นแบบ personal WWW proxy คือ ใช้สำหรับเป็นตัวกลางรับ/ส่งข้อมูลสำหรับเครื่อง client เพียงเครื่องเดียว โดยจะเป็นโปรแกรมที่ทำงานอยู่บนเครื่องของ client เอง ในการเก็บข้อมูลนั้นจะอาศัย

เนื้อที่ใน hard disk หรือหน่วยความจำ หรือทั้งสองอย่างในการเก็บข้อมูลที่เคยมีการเรียกใช้มาแล้ว ซึ่งวัตถุประสงค์ของการใช้งาน personal WWW proxy นั้น อาศัยสมมติฐานที่ว่า ผู้ใช้คนหนึ่งๆ จะมีความเป็นไปได้สูงที่จะเรียกดู web site ที่ซ้ำๆกันทุกวัน หรือวันละหลายๆครั้ง จึงควรมีการ cache ข้อมูลเหล่านั้นไว้ที่เครื่อง client เอง เพื่อเพิ่มความเร็วในการเข้าถึงข้อมูลที่ซ้ำๆกัน

นอกจากนี้ การใช้งาน personal WWW proxy ยังสามารถนำไปใช้ในการกรองข้อมูลที่ไม่ต้องการให้มีการเรียกใช้งาน เช่น web site ที่มีข้อมูลที่ไม่เหมาะสมสำหรับเด็ก, แถบโฆษณาต่างๆ เป็นต้น

4.3 การทำงานของ WWW proxy

ขั้นตอนการทำงานของโปรแกรม WWW proxy นั้นสามารถกล่าวโดยสรุปได้ดังนี้

1. WWW proxy เริ่มการเปิดให้บริการโดยกำหนดค่าหมายเลข port ที่เปิดให้บริการ, ข้อมูลของ gateway ที่ใช้ ฯลฯ

2. เริ่มต้นการรอรับการร้องขอจาก client (เช่น WWW browser) โดยทำการสร้างช่องทางในการติดต่อ เพื่อรอรับการติดต่อจาก client ทาง Internet socket แบบ stream socket ที่เปิดให้บริการที่หมายเลข port ที่ระบุตามข้อ 1

3. เมื่อมีการร้องขอการเชื่อมต่อ (connection) จาก client มายังหมายเลข port ตามข้อ 1 โปรแกรม WWW proxy จะยอมรับการเชื่อมต่อและเมื่อทำการเชื่อมต่อระหว่าง WWW proxy กับ client แล้ว โปรแกรม WWW proxy จะรอรับ HTTP request ที่จะส่งมาจาก client สำหรับทรัพยากร (เอกสาร html หรือข้อมูลอื่นๆ) ที่ต้องการ เพื่อให้ WWW proxy ส่งต่อ HTTP request นั้น ไปยัง WWW server ต่อไป

โดยลักษณะของ HTTP request ที่ user agent ส่งให้กับ WWW proxy (กรณีติดต่อผ่านทาง WWW proxy) นั้น จะแตกต่างจาก HTTP request ที่ client ส่งให้กับ WWW server โดยตรง (กรณีที่ไม่ได้ติดต่อผ่าน WWW proxy) คือ HTTP request ที่ส่งให้กับ WWW proxy นั้นในส่วนของ request URL จะประกอบด้วย ชื่อของ WWW server ที่ต้องการติดต่อ, path ที่อยู่ของทรัพยากร และชื่อของทรัพยากรที่ต้องการนั้นๆ (เช่น "GET http://www.w3.org/pub/www/TheProject.html HTTP/1.0" เป็นต้น) แต่ใน HTTP request ที่ส่งให้กับ WWW server โดยตรงนั้น ในส่วนของ request URL จะประกอบด้วย path ที่อยู่ของทรัพยากร และชื่อของทรัพยากรที่ต้องการนั้นๆเท่านั้น (เช่น "GET /pub/www/ Theproject.html HTTP/1.0" เป็นต้น) เนื่องจาก WWW proxy เป็นการให้บริการการเชื่อมต่อกับ WWW server อีกทอดหนึ่ง ดังนั้น user agent ที่ใช้บริการ WWW proxy จึงต้องระบุชื่อของ WWW server ที่เป็นเจ้าของทรัพยากรนั้นๆด้วย แต่ในกรณีที่ user agent ติดต่อกับ

WWW server โดยตรง user agent สามารถส่ง HTTP request ไปยัง WWW server ผ่านทางการเชื่อมต่อกับ server ที่สร้างไว้โดยตรงอยู่แล้ว

ในขั้นตอนนี้จะเห็นว่า โปรแกรม WWW proxy ทำงานในลักษณะที่เป็นเสมือน WWW server ผู้ให้บริการทรัพยากรให้แก่ client เอง แต่จะมีความแตกต่างเพียงส่วนของ request URL ที่ client ส่งมาเท่านั้น

4. โปรแกรม WWW proxy ทำการแยกส่วนต่างๆของ HTTP request นั้นออกเป็น 2 ส่วน หลักๆคือ ส่วนของชื่อของ WWW server ที่ client ต้องการติดต่อ และส่วนของ path ที่อยู่และชื่อของทรัพยากรนั้นๆ เพื่อสร้างเป็น HTTP request ใหม่ที่จะส่งไปยัง WWW server (HTTP request ที่สร้างขึ้นใหม่นี้จะมีความแตกต่างไปจากเดิมคือ จะไม่มีส่วนของ ชื่อของ WWW server ประกอบอยู่ และอาจมีการตัด HTTP header อื่นๆที่ไม่จำเป็นออก เช่น ข้อมูลของ client ที่ระบุอยู่ใน request header เป็นต้น)

ตัวอย่างเช่น เมื่อ user agent ส่ง HTTP request มายัง WWW proxy เป็นดังนี้

"GET http://www.w3.org/pub/www/TheProject.html HTTP/1.0"

เมื่อโปรแกรม WWW proxy ได้รับ request message นี้แล้ว ก็จะทำการแยกส่วน message ออกเป็น 2 ส่วนคือ ส่วนของชื่อของ WWW server ซึ่งก็คือ "www.w3.org" เพื่อใช้ในการสร้างการเชื่อมต่อ (connection) ต่อไปยัง WWW server นั้น

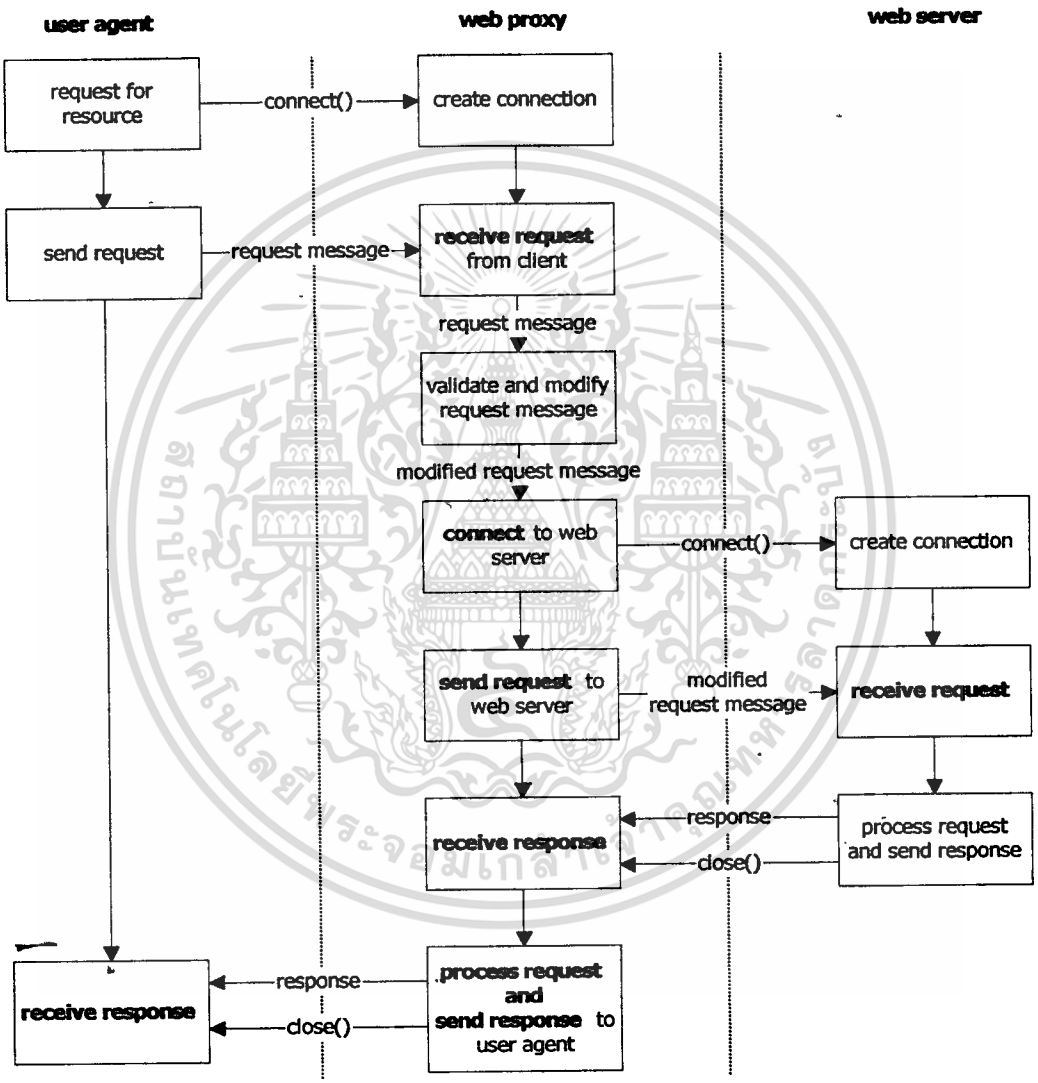
และส่วนของ path ที่อยู่และชื่อของทรัพยากร ซึ่งก็คือ "/pub/www/TheProject.html" เพื่อสร้างเป็น HTTP request ใหม่ที่จะส่งไปยัง WWW server "www.w3.org" ต่อไป โดยจะได้ HTTP request ใหม่เป็น "GET /pub/www/ Theproject.html HTTP/1.0"

5. หลังจากนั้นโปรแกรม WWW proxy จะทำการสร้างช่องทางสำหรับการเชื่อมต่อ (socket) อีกช่องทางหนึ่ง เพื่อใช้ในการติดต่อไปยัง WWW server และทำการเชื่อมต่อกับ WWW server ผ่านทางช่องทางการเชื่อมต่อนั้นๆ และส่ง HTTP request ที่สร้างขึ้นมาใหม่ให้กับ WWW server

ในขั้นตอนนี้ จะเห็นว่าโปรแกรม WWW proxy จะทำงานเสมือนเป็น user agent ตัวหนึ่งที่ร้องขอทรัพยากรจาก WWW server โดยที่ WWW server ไม่จำเป็นต้องทราบเลยว่าเป็นการร้องขอจาก WWW proxy ไม่ใช่ user agent จริงๆ

6. เมื่อ WWW server ได้รับ HTTP request จาก WWW proxy แล้ว ก็จะจัดการตรวจสอบสิทธิการเข้าใช้ทรัพยากร และส่งตอบสถานะของการประมวลผลและข้อมูลที่ร้องขอ (ถ้ามี) กลับไปให้กับ WWW proxy โดยอยู่ในรูปของ HTTP response

7. เมื่อ WWW proxy ได้รับ HTTP response ที่ตอบกลับมาจาก WWW server แล้ว จะทำการตรวจสอบสถานะของ response ที่ได้รับ และส่งข้อความแสดงข้อผิดพลาด หรือส่งข้อมูลทรัพยากรที่ร้องขอ กลับไปให้กับ client อีกทอดหนึ่ง จากนั้นจึงปิดการเชื่อมต่อที่สร้างไว้ทั้งกับ client และกับ WWW server



รูปที่ 4.2 แสดงลักษณะการทำงานของโปรแกรม web proxy

จากการที่ WWW proxy เป็นตัวกลางในการติดต่อสื่อสารระหว่าง web client กับ WWW server ดังที่ได้กล่าวมาแล้ว จะเห็นว่าเราสามารถนำเอาข้อมูลทรัพยากรต่างๆที่ WWW server ส่งให้กับ client ผ่านทาง WWW proxy นี้ไปใช้ประโยชน์อื่นๆได้ นอกเหนือจากการ caching หรือการส่งกลับไปแสดงผลยัง client อย่างเดียว ตัวอย่างที่เห็นในปัจจุบัน เช่น ได้มีการพัฒนาโปรแกรมที่ใช้

หลักการนี้เพื่อใช้ในจำกัดการเข้าถึง web site ที่ไม่ต้องการ เช่น web site ลามก, web site โฆษณาสินค้า รวมไปถึงการตัดเอาแถบโฆษณาสินค้าแบบ popup ออกจากหน้า web page เป็นต้น

ในบทถัดไป จะกล่าวถึงการศึกษาและพัฒนาปรับปรุงโปรแกรม WWW proxy ที่สามารถนำข้อมูลทรัพยากรเหล่านั้นไปประมวลผลเพื่อใช้ประโยชน์ในทางต่างๆ ตามแนวคิดที่กล่าวมา

บทที่ 5
การพัฒนา WWW proxy ที่สามารถนำข้อมูล WWW ไปประมวลผล

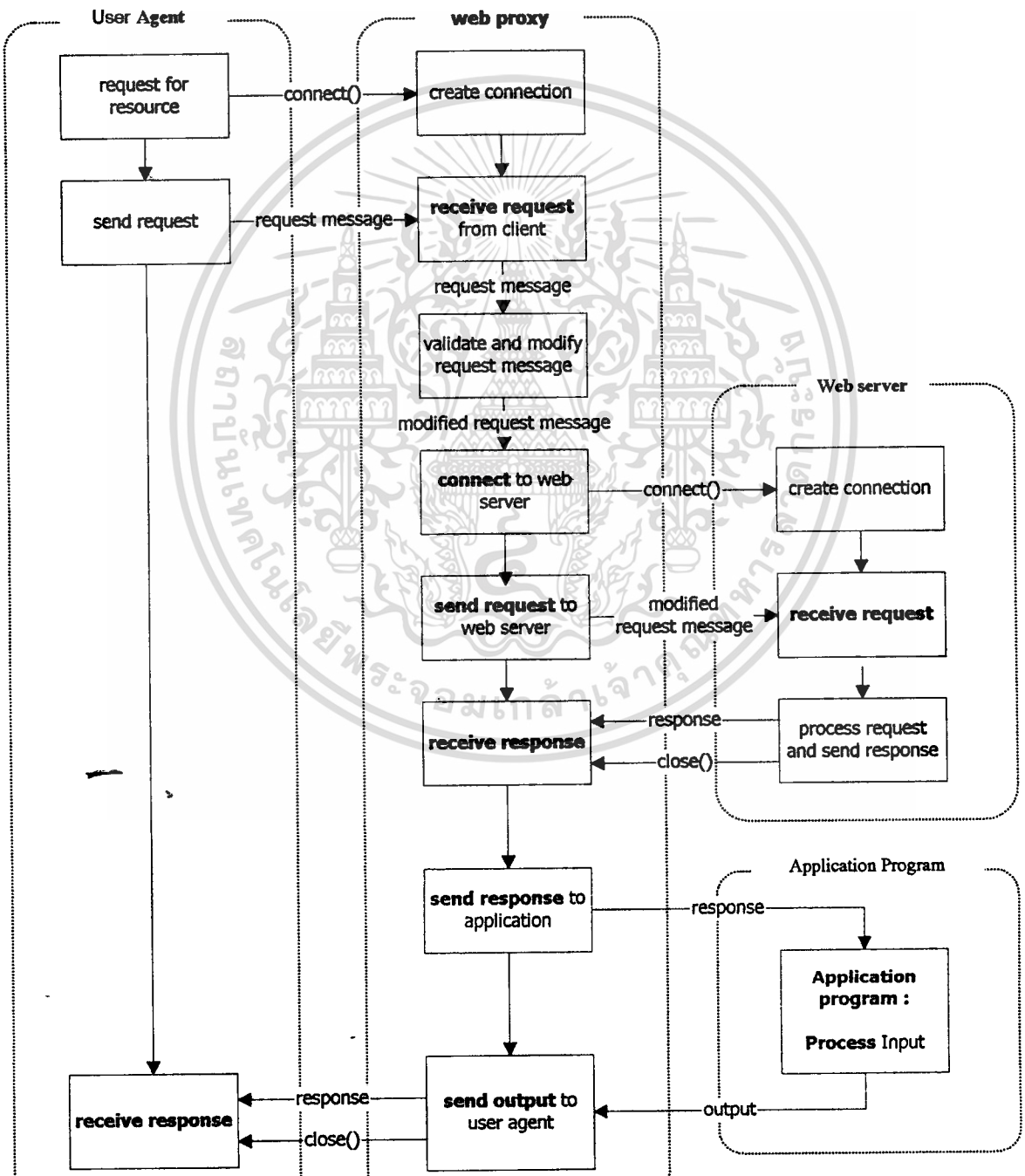
ในบทนี้จะกล่าวถึงการศึกษาพัฒนาโปรแกรม WWW proxy เพื่อปรับปรุงให้สามารถนำข้อมูลทรัพยากร World-Wide Web ที่ร้องขอโดย client ผ่านทาง WWW proxy ไปยัง WWW server มาใช้เพื่อประมวลผลเพื่อใช้ประโยชน์ ซึ่งโปรแกรมที่พัฒนาขึ้นในครั้งนี้มีรายละเอียดดังนี้

- เครื่องมือที่ใช้ในการพัฒนาโปรแกรม :	- กระทำโดยใช้ภาษาโปรแกรม Visual C++ version 6.0 โดยอาศัยชุดเครื่องมือพัฒนาโปรแกรม MFC (Microsoft Fundamental Class)
- ความต้องการพื้นฐาน :	- เครื่อง PC หน่วยประมวลผล Intel Pentium Compatible - หน่วยความจำหลัก (RAM) 16 MB.
- ระบบปฏิบัติการ :	Microsoft Windows 98

ในการทำงานนั้น สามารถแยกออกเป็น 4 ส่วน ดังรูปที่ 6.1 ได้แก่

1. User Agent ซึ่งได้แก่ โปรแกรม WWW browser ต่างๆที่ร้องขอทรัพยากร World-Wide Web
2. WWW proxy ที่ได้พัฒนาขึ้นในโครงการนี้ โดยจะมีการเพิ่มเติมในส่วนของการรับ/ส่งข้อมูล World-Wide Web ที่ได้รับจาก WWW server ไปยัง application program ที่รอรับข้อมูล และทำการประมวลผลข้อมูลนั้น ตามแต่ลักษณะของแต่ละ application จากนั้นจึงส่งผลลัพธ์กลับไปยัง WWW proxy
3. Application program เป็นโปรแกรมที่พัฒนาขึ้นเพิ่มเติมเพื่อทำหน้าที่ในการประมวลผลข้อมูล World-Wide Web ที่ได้รับจาก WWW proxy
4. WWW server ผู้ให้บริการ World-Wide Web

ทั้งนี้ ในการนำข้อมูล WWW ที่ได้รับจาก WWW server ไปประมวลผลของ WWW proxy นั้น จะกระทำโดย ส่งข้อมูลเหล่านั้นไปให้กับ application program อีกโปรแกรมหนึ่ง ที่พัฒนาขึ้นเพื่อทำการประมวลผลเฉพาะอย่าง แทนที่จะทำการประมวลผลข้อมูลเหล่านั้นโดยตัว WWW proxy เอง เพื่อให้สามารถเลือกใช้ application program ในการประมวลผลแบบต่างๆได้ โดยไม่ต้องแก้ไขโปรแกรม WWW proxy



รูปที่ 5.1 การส่งข้อมูลของโปรแกรม web proxy ไปประมวลผลยัง application program

5.1 การพัฒนาโปรแกรม

ในการพัฒนาโปรแกรม WWW proxy ขึ้น ใช้ภาษาโปรแกรม Visual C++ version 6.0 ซึ่งเป็นภาษาพัฒนาโปรแกรมเชิงวัตถุ (Object-Oriented Programming Language) โดยอาศัยชุดเครื่องมือพัฒนาโปรแกรมของ MFC (Microsoft Fundamental Class) เป็นหลัก โดยในส่วนของการทำงานของการให้บริการเป็นตัวกลางในการติดต่อสื่อสารนั้น อาศัยขั้นตอนการทำงานดังที่กล่าวในหัวข้อ 4.3 ทั้งนี้ได้มีสร้าง object ต่างๆเพื่อใช้ในการทำงานของโปรแกรม ดังนี้

1. Class CClientState เป็น class หลักในการทำงานสำหรับเก็บข้อมูลของการติดต่อระหว่างที่มีการติดต่อกันของโปรแกรม WWW proxy กับ WWW browser (client) และระหว่างโปรแกรม WWW proxy กับ WWW server

```
class CClientState
{
public:
        void ParseHttpRequest(char* req);
        char* GetHeader();
        CClientState();
        virtual ~CClientState();

        CBlockingSocket  cfd;
        CBlockingSocket  sfd;
        char *ip_addr_str;
        long  ip_addr_long;
        Ciob iob;
        struct http_request http[1];
};
```

รูปที่ 5.2 แสดง class CClientState

โดยจะประกอบด้วยตัวแปรสมาชิก (member variable) ดังนี้

- *cfd* เป็น Object ชนิด CBlockingSocket สำหรับเป็น socket ที่จะสร้างขึ้นและใช้งานในการติดต่อระหว่างโปรแกรม WWW proxy กับ WWW browser
- *sfd* เป็น Object ชนิด CBlockingSocket สำหรับเป็น socket ที่จะสร้างขึ้นและใช้งานในการติดต่อระหว่างโปรแกรม WWW proxy กับ WWW server
- ตัวแปรชนิด char * *Ip_addr_str* สำหรับเก็บค่าของ IP address ของ client ที่ทำการติดต่อในรูปแบบของ string

- ตัวแปรชนิด long *Ip_addr_long* สำหรับเก็บค่าของ IP address ของ client ที่ทำการติดต่อในรูปแบบของ long
- *iob* เป็น Object ชนิด Ciob ใช้สำหรับเป็น input buffer เก็บข้อความ HTTP request ที่ client ส่งเข้ามา
- ตัวแปรโครงสร้างชนิด struct http_request ใช้สำหรับเก็บข้อความ HTTP request ที่ client ส่งเข้ามา แต่ได้รับการแยกส่วนต่างๆของ HTTP request ออกแล้ว

และประกอบด้วย method ต่างๆดังนี้

- method *ParseHttpRequest()* ใช้ในการแยกส่วนต่างๆของ HTTP request ที่ client ส่งเข้ามา โดยจะแยกออกเป็นส่วนของ HTTP header, HTTP method, URI path และ address ของ WWW server เพื่อใช้ในการสร้างเป็น HTTP request ใหม่เพื่อส่งให้กับ WWW server ต่อไป
- method *GetHeader()* ใช้ในการค้นหา HTTP header ของ HTTP request ที่ส่งมาจาก client

2. Class Ciob เป็น class สำหรับเก็บข้อความ HTTP request ที่ส่งเข้ามาจาก client

```
class Ciob
{
public:
    Ciob();
    virtual ~Ciob();
    int add_iob(char* buffer, int len);
    void Clear();

    int size;
    char* buf;
    char* cur;
    char* eod;
};
```

รูปที่ 5.3 แสดง class Ciob

ประกอบด้วย

- ตัวแปรชนิด int *size* สำหรับระบุขนาดความยาวของข้อความ HTTP request
- ตัวแปรชนิด char* *buf* สำหรับเก็บข้อความ HTTP request

- ตัวแปรชนิด char* *cur* สำหรับระบุตำแหน่งปัจจุบันในข้อความ buf ที่ทำการประมวลผลอยู่
- ตัวแปรชนิด char* *eod* สำหรับระบุตำแหน่งสุดท้ายของข้อความ buf
- method *add_iob()* ใช้สำหรับทำการเก็บข้อมูล HTTP request (buffer) ไปเก็บไว้ใน object Ciob
- method *Clear()* ใช้สำหรับทำการลบข้อมูลทั้งหมดที่อยู่ใน object Ciob

3. **ตัวแปรโครงสร้าง http_request** เป็นตัวแปรที่ใช้เก็บข้อความ HTTP request ที่ client ส่งเข้ามา ที่ได้ถูกแยกออกเป็นส่วนต่างๆแล้ว

```
struct http_request
{
        char *cmd;
        char *gpc;
        char *host;
        int   port;
        char *path;
        char *ver;
        char *hostport;
};
```

รูปที่ 5.4 แสดงตัวแปรโครงสร้าง http_request

ประกอบด้วย

- *cmd* เป็นส่วนของ request line ของ HTTP request
- *gpc* เป็นส่วนของ request method ของ HTTP request
- *host* เป็นส่วนของชื่อ WWW server ที่ client ต้องการติดต่อ
- *port* เป็นหมายเลข port ของ server ที่ client ต้องการติดต่อ (โดยปกติแล้วจะเป็น 80)
- *path* เป็น path ของทรัพยากรที่ client ร้องขอจาก WWW server
- *ver* เป็น HTTP version ของ HTTP request ที่ client ส่งเข้ามา
- *hostport* เป็นข้อความ ชื่อ WWW server และ หมายเลข port ที่ client ต้องการติดต่อ

4. Class CBlockingSocket เป็น class ที่สร้างขึ้นจากฟังก์ชันการใช้งาน Internet socket พื้นฐานต่างๆ (ใน socket API) เพื่อใช้ในการติดต่อสื่อสารระหว่างโปรแกรม WWW proxy กับ client และ WWW server โดยการสร้างเป็น Object ของ socket แทนการใช้งาน socket API ตามปกติ

```
typedef const struct sockaddr* LPCSOCKADDR;

class CBlockingSocket : public CObject
{
        DECLARE_DYNAMIC(CBlockingSocket)
public:
        SOCKET m_hSocket;
        CBlockingSocket() { m_hSocket = NULL; }
        void Cleanup();
        void Create(int nType = SOCK_STREAM);
        void Close();
        void Bind(LPCSOCKADDR psa);
        void Listen();
        void Connect(LPCSOCKADDR psa);
        void Connect(char* hostname);
        BOOL Accept(CBlockingSocket& s, LPSOCKADDR psa);
        int Send(const char* pch, const int nSize,
              const int nSecs);
        int Write(const char* pch, const int nSize,
              const int nSecs);
        int Receive(char* pch, const int nSize,
              const int nSecs);
        int SendDatagram(const char* pch, const int nSize,
              LPCSOCKADDR psa, const int nSecs);
        int ReceiveDatagram(char* pch, const int nSize,
              LPSOCKADDR psa, const int nSecs);
        void GetPeerAddr(LPSOCKADDR psa);
        void GetSockAddr(LPSOCKADDR psa);
}
```

รูปที่ 5.5 แสดง class CBlockingSocket

ประกอบด้วย ตัวแปรสมาชิก และ method ต่างๆที่ทำงานเช่นเดียวกับการทำงานของฟังก์ชันของ socket API

- ตัวแปรชนิด SOCKET *m_hSocket* เป็น Socket Handler ใช้ระบุถึง Internet socket ที่สร้างขึ้นและใช้งานในการติดต่อสื่อสาร
- method *CBlockingSocket()* เป็น constructor method สำหรับ class CblockingSocket

- method `Cleanup()` ใช้ในการยกเลิกการใช้งาน socket ที่สร้างขึ้นจาก method `Create()`
- method `Create()` ใช้ในการสร้าง Internet socket เพื่อใช้ในการสื่อสาร โดยระบุชนิดของ socket ที่ต้องการสร้างในพารามิเตอร์ `nType` (SOCK_STREAM สำหรับ stream socket หรือ SOCK_DGRAM สำหรับ datagram socket)
- method `Close()` ใช้ในการยกเลิกการใช้งาน socket ที่สร้างขึ้นจาก method `Create()`
- method `Bind()` ใช้ในการกำหนด address อ้างอิงให้กับ socket `m_hSocket`
- method `Listen()` ใช้ในการกำหนดให้ socket `m_hsocket` เข้าสู่สถานะ listen เพื่อรอรับการเชื่อมต่อจาก client
- method `Connect()` ใช้ในการขอสร้างการเชื่อมต่อไปยัง socket ของ server อื่นๆ
- method `Accept()` ใช้สำหรับ server ในการยอมรับการร้องขอสร้างการเชื่อมต่อจาก client ผ่านทาง socket ที่สร้างขึ้น และสร้างการเชื่อมต่อระหว่าง socket ของ client กับ object CBlockingSocket `s` ที่สร้างขึ้นใหม่ และคืนค่ากลับมา
- method `Send()` ทำการส่งข้อมูลแบบ stream ไปยังผู้รับซึ่งอยู่ฝ่ายตรงข้ามของการเชื่อมต่อ ของ stream socket (ใช้โปรโตคอล TCP)
- method `Receive()` ใช้ในการรับข้อมูลจากผู้ส่งซึ่งอยู่ฝ่ายตรงข้ามของการเชื่อมต่อของ stream socket (ใช้โปรโตคอล TCP)
- method `SendDatagram()` ใช้ในการส่งข้อมูลไปยังผู้รับตามที่ระบุใน address อ้างอิง psa ด้วย datagram socket (ใช้โปรโตคอล UPD)
- method `ReceiveDatagram()` ใช้ในการรับข้อมูลที่ส่งเข้ามายัง socket ที่สร้างขึ้น (ใช้โปรโตคอล UDP)
- method `GetPeerAddr()` ใช้ในการหาค่า address อ้างอิงของอีกฝ่ายหนึ่งของการเชื่อมต่อ (ใน stream socket ที่ทำการเชื่อมต่อแล้ว) และคืนกลับค่า address นั้นทาง ตัวแปร `psa`
- method `GetSockAddr()` ใช้ในการหาค่า address อ้างอิง socket m_hSocket และคืนกลับค่า address นั้นทาง ตัวแปร `psa`

5. Class CsockAddr เป็น class ที่สร้างขึ้นจากตัวแปรโครงสร้าง socket address พื้นฐานของ Internet socket เพื่อให้เกิดความสะดวกในการใช้งานตัวแปรโครงสร้าง socket address มากยิ่งขึ้น ประกอบด้วย method ต่างๆ ที่ใช้ในการเรียกใช้ข้อมูลภายในตัวแปรโครงสร้าง socket address

```
class CSockAddr : public sockaddr_in {
public:
   CSockAddr()
          { sin_family = AF_INET;
            sin_port = 0;
            sin_addr.s_addr = 0; }
   CSockAddr(const SOCKADDR& sa) { memcpy(this, &sa,
          sizeof(SOCKADDR)); }
   CSockAddr(const SOCKADDR_IN& sin) { memcpy(this, &sin,
          sizeof(SOCKADDR_IN)); }
   CSockAddr(const ULONG ulAddr, const USHORT ushPort = 0)
          { sin_family = AF_INET;
            sin_port = htons(ushPort);
            sin_addr.s_addr = htonl(ulAddr); }
   CSockAddr(const char* pchIP, const USHORT ushPort = 0)
          { sin_family = AF_INET;
            sin_port = htons(ushPort);
            sin_addr.s_addr = inet_addr(pchIP); }

   CString DottedDecimal()
          { return inet_ntoa(sin_addr); }
   USHORT Port() const
          { return ntohs(sin_port); }
   ULONG IPAddr() const
          { return ntohl(sin_addr.s_addr); }
};
```

รูปที่ 5.6 แสดง class CSockAddr

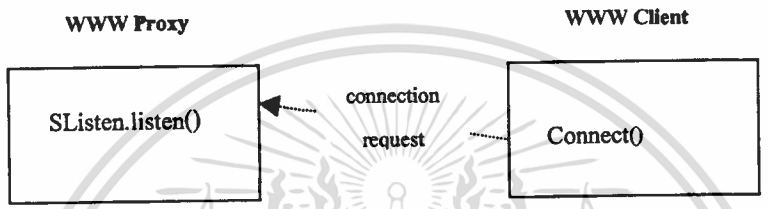
ประกอบด้วย method ต่างๆดังนี้

- Constructor รูปแบบต่างๆ สำหรับสร้าง object ของ socket address โดยระบุพารามิเตอร์ที่แตกต่างกัน
- method *DotDecimal()* เป็น method ที่คืนกลับค่าของ IP address ที่เก็บอยู่ในตัวแปรโครงสร้าง socket address ในรูปแบบของ dotted decimal (XXX.XXX.XXX.XXX)
- method *Port()* เป็น method ที่คืนกลับค่าของหมายเลข port ที่เก็บอยู่ในตัวแปรโครงสร้าง socket address
- method *IPAddr()* เป็น method ที่คืนกลับค่าของ IP address ที่เก็บอยู่ในตัวแปรโครงสร้าง socket address

5.2 การทำงานของโปรแกรม WWW proxy ในบริการหลายไคลเอนท์พร้อมกัน

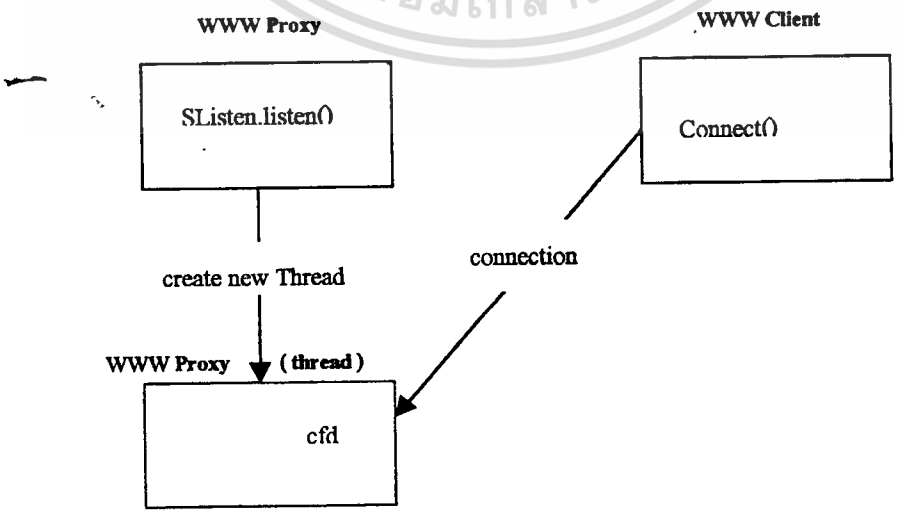
ในการทำงานของโปรแกรม WWW proxy นั้น จะทำในลักษณะของ multithread เพื่อให้สามารถให้บริการต่อ client ที่ร้องขอเข้ามาหลาๆ client พร้อมๆกัน โดยไม่จำเป็นต้องรอการทำงานของ client อื่นๆเสร็จสิ้นเสียก่อน โดยเริ่มจาก

1. โปรแกรม WWW proxy ทำการสร้าง object ของ CBlockingSocket หลักขึ้นและใช้ในการรอรับการเชื่อมต่อจาก client (*SListen*)



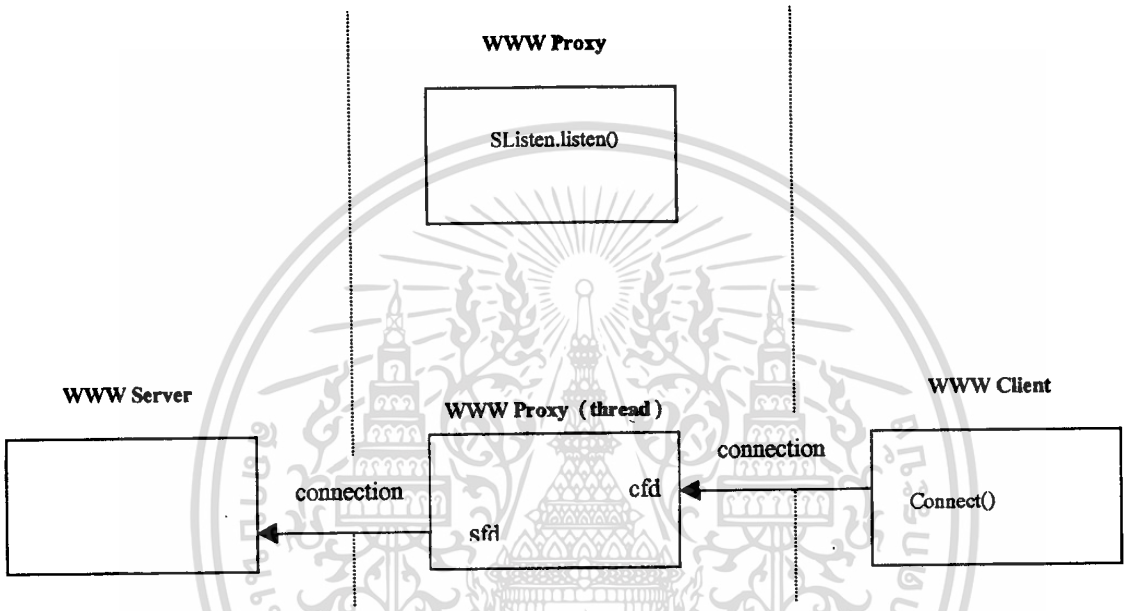
รูปที่ 5.7 WWW proxy และ WWW client ก่อนการสร้าง connection

2. เมื่อมี client ร้องขอสร้างการเชื่อมต่อเข้ามายัง WWW proxy ผ่านทาง object ที่สร้างไว้ก็จะทำการสร้าง thread ใหม่เพื่อให้บริการแก่แต่ละ client นั้นๆ โดยที่ในแต่ละ thread จะทำการสร้าง object ของ CClientState เพื่อใช้ในการเก็บข้อมูลและการทำงานต่างๆ ระหว่างการให้บริการแก่ client นั้นๆ โดยจะประกอบด้วยส่วนสำคัญคือ object ของ CBlockingSocket 2 อัน สำหรับเชื่อมต่อระหว่างโปรแกรม WWW proxy กับ client (object *cfd*) และสำหรับเชื่อมต่อระหว่าง WWW server กับโปรแกรม WWW proxy (object sfd)



รูปที่ 5.8 WWW proxy และ WWW client หลังการสร้าง connection

3. ทำการเชื่อมต่อระหว่างโปรแกรม WWW proxy กับ client โดยใช้ socket object cfd
4. รับข้อความ HTTP request ที่ ส่งมาจาก client และให้บริการแก่ request นั้นๆตามวิธีการทำงานของโปรแกรม proxy (ดังหัวข้อ 4.3 (4) – (7)) ทั้งนี้การเชื่อมต่อกับ WWW server จะกระทำโดยใช้ socket object sfd ของแต่ละ thread นั้นๆ



รูปที่ 5.9 ลักษณะ connection ระหว่าง WWW proxy, WWW client และ WWW server

5. ในขณะที่แต่ละ thread ทำการให้บริการแก่แต่ละ client นั้น โปรแกรม WWW proxy จะสามารถรอรับการร้องขอจาก client อื่นๆที่อาจมีเข้ามา และทำการสร้าง thread ใหม่เพื่อให้บริการแก่ client นั้น ได้ตลอดเวลา
6. หลังจากเสร็จสิ้นการให้บริการแก่ HTTP request ของ client นั้นๆแล้ว โปรแกรมก็จะยุติการเชื่อมต่อทั้งกับ client และ WWW server และจบการทำงานของ thread นั้น

5.2.1 การสร้าง thread โดยฟังก์ชัน AfxBeginThread()

ในการทำงานของโปรแกรมแบบ multithread นั้น แต่ละ thread ใน process จะใช้เนื้อที่หน่วยความจำร่วมกัน รวมถึงการใช้ทรัพยากรอื่นๆร่วมกันด้วย ดังนั้นการใช้ข้อมูลร่วมกัน และการติดต่อกันระหว่าง thread จึงเป็นไปได้ง่าย โดยแต่ละ thread จะมีหมายเลข thread (thread ID), register และ program counter, ตัวแปร local เป็นของตนเอง ในการทำงานนั้น CPU ไม่จำเป็นต้องทำ context switching เพื่อสลับเปลี่ยนการทำงานสำหรับแต่ละ thread แต่ CPU เพียงแค่สลับข้อมูล

ที่เก็บอยู่ใน register, program counter และ stack สำหรับแต่ละ thread เท่านั้น ซึ่งสะดวกและรวดเร็วมาก แต่เนื่องจากการทำงานโดยใช้หน่วยความจำร่วมกัน การใช้ข้อมูลร่วมกันอาจทำให้เกิดการไม่ถูกต้องตรงกันของข้อมูลในแต่ละ thread ได้

การพัฒนาโปรแกรมในการทำงานแบบ multithread โยภาษาโปรแกรม Visual C++ นั้น สามารถทำได้โดยการสร้างฟังก์ชันการทำงานหลักของ thread ซึ่งจะต้องคืนค่ากลับเป็น *UINT* และรับค่า *LPVOID* เป็นพารามิเตอร์ สำหรับเป็นฟังก์ชันที่แต่ละ thread ที่จะสร้างขึ้น ทำงานเป็นอันดับแรก ตัวอย่างเช่น

```
UNIT ComputeThreadProc(LPVOID pParam)
{
      .....
      return 0;
}
```

การสร้าง thread ใหม่เพื่อใช้ในการทำงานสามารถทำได้โดยฟังก์ชัน `AfxBeginThread()` โดยมีรูปแบบดังนี้

```
CWinThread* AfxBeginThread(
      AFX_THREADPROC pfnThreadProc,
      LPVOID pParam,
      int nPriority = THREAD_PRIORITY_NORMAL,
      UINT nStackSize = 0,
      DWORD dwCreateFlags = 0,
      LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL
);
```

pfnThreadProc เป็นชื่อของฟังก์ชันหลักที่ต้องการให้ thread ที่สร้างขึ้นเริ่มทำงาน

pParam เป็นพารามิเตอร์ที่ต้องการส่งให้กับ ฟังก์ชัน *pfnThreadProc()* ทั้งนี้ในการส่งค่าพารามิเตอร์หลายๆตัวให้กับฟังก์ชัน ทำได้โดยการสร้างตัวแปรโครงสร้าง (structure) ขึ้นสำหรับเก็บค่าพารามิเตอร์ทั้งหมด และทำการส่ง address ของตัวแปรโครงสร้างนั้น เป็นพารามิเตอร์ให้กับ `AfxBeginThread()` ในส่วนของ pParam นี้ เป็นแบบ void *

nPriority เป็นค่าความสำคัญในการประมวลผล (priority) ที่ต้องการกำหนดให้กับ thread โดยหากไม่มีการระบุ ค่า *nPriority* จะมีค่าเป็น `THREAD_PRIORITY_NORMAL`

nStackSize เป็นขนาดของ stack หน่วยเป็น Byte ที่ต้องการกำหนดให้กับ thread เพื่อใช้ในการประมวลผล หากไม่มีการระบุ ค่าของ *nStackSize* จะมีค่าเป็น 0 นั่นคือให้ขนาดของ stack มีค่าเท่ากับขนาดของ stack ของ thread ที่เป็นผู้สร้าง thread นี้ขึ้น

dwCreateFlags เป็นค่าสถานะควบคุมพฤติกรรมของการสร้าง thread จะมีค่าได้เป็น

- `CREATE_SUSPEND` จะทำให้ thread ที่สร้างขึ้นอยู่ในสถานะ SUSPEND (หยุดทำงาน) และจะเริ่มทำงานก็ต่อเมื่อมีการเรียกใช้ฟังก์ชัน `ResumeThread()`
- 0 จะทำให้ thread ที่สร้างขึ้นใหม่เริ่มทำงานในฟังก์ชัน *pfnThreadProc* () ทันที

lpSecurityAttrs ใช้สำหรับการกำหนดค่าความปลอดภัย (security) ของ thread ที่สร้างขึ้น หากไม่มีการระบุ จะมีค่าเป็น NULL นั่นคือ ไม่มีการกำหนดค่าใดๆ

ตัวอย่างการทำงานแบบ mutithread (บางส่วน) ของโปรแกรม WWW proxy

```
1   CBlockingSocket sListen;

2   void CProxyDoc::StartServer()
3   {
4           .....
5           try {
6                   CSockAddr sServer(INADDR_ANY, 8000);
7                   sListen.Create();
8                   sListen.Bind(sServer);
9                   sListen.Listen();
10                  AfxBeginThread(ServerThreadProc,
                            THREAD_PRIORITY_NORMAL);
11          }
12          catch(CBlockingSocketException* e) {
13                  .....
14          }
15  }
```

รูปที่ 5.10 ตัวอย่างการทำงานแบบ mutithread (บางส่วน) ของโปรแกรม

บรรทัดที่ (1) เป็นการกำหนด object CblockingSocket `sListen` เพื่อใช้เป็น socket ในการรอรับการขอเชื่อมต่อจาก client (listen)

method `CProxyDoc::StartServer()` เป็น method ของ object CproxyDoc ทำหน้าที่เริ่มการทำงานของ WWW proxy คือ สร้าง socket สำหรับรอรับการเชื่อมต่อ โดยเรียกใช้ method `sListen.Create()` (บรรทัดที่ 7) จากนั้นทำการกำหนด address อ้างอิง (`sListen.Bind()` บรรทัดที่ 8) และรอรับการขอเชื่อมต่อจาก server (`sListen.Listen()` บรรทัดที่ 9)

จากนั้น ทำการสร้าง thread ใหม่ โดยระบุฟังก์ชันแรกของการทำงานของ thread เป็น `ServerThreadProc()` เพื่อให้แต่ละ thread ทำการเชื่อมต่อและให้บริการแก่ client ที่ติดต่อเข้ามาแต่ละ client

```
16  UINT ServerThreadProc(LPVOID pParam)
17  {
18   CSockAddr    saClient;
19   CClientState client;

20         try {
21               if(!sListen.Accept(client.cfd, saClient)) {
22                     return 0;
23               }
24               AfxBeginThread(ServerThreadProc,pParam);
25               client.ip_addr_str =
                       strdup(inet_ntoa(saClient.sin_addr));
26               client.ip_addr_long =
                       ntohl(saClient.sin_addr.s_addr);

27               serve(&client);
28               client.cfd.Close();
29               client.sfd.Close();
30         }
31         catch (CBlockingSocketException* e) {
32               .....
33         }
34   return 1;
35  }

36  void serve(CClientState *csp)
37  {
38         .....
39  }
```

รูปที่ 5.11 ตัวอย่างการทำงานแบบ mutithread (บางส่วน) ของโปรแกรม (ต่อ)

ฟังก์ชัน `ServerThreadProc()` เป็นฟังก์ชันการทำงานหลักของแต่ละ thread ที่สร้างขึ้น โดยมีการสร้าง local object CClientState `client` ขึ้นเพื่อใช้สำหรับการติดต่อระหว่าง thread นั้นๆกับ client และ WWW server (บรรทัดที่ 19)

จากนั้นทำการสร้างการเชื่อมต่อกับ client ที่ได้ร้องขอเข้ามาทาง socket ที่รอรับการขอเชื่อมต่ออยู่ (`sListen`) โดยสร้างการเชื่อมต่อเข้ากับ socket ที่สร้างขึ้นใหม่ของ object `client` (`client.cfd` บรรทัดที่ 21)

ในบรรทัดที่ 22 ทำการสร้าง thread ใหม่ขึ้นอีก thread หนึ่งเพื่อใช้เชื่อมต่อกับ client อื่นๆที่อาจจะติดต่อเข้ามาทาง `sListen` ต่อไป

จากนั้น ก็จะทำการให้บริการแก่ client โดยฟังก์ชัน `serve()` (บรรทัดที่ 27) โดยจะเป็นการรับข้อความ HTTP request จาก client ผ่านทาง socket `client.cfd` ที่ได้สร้างการเชื่อมต่อไว้แล้ว , ติดต่อไปยัง WWW server เพื่อขอใช้ทรัพยากรที่ client ต้องการ ผ่านทาง socket `client.sfd` และส่งผลลัพธ์กลับไปให้กับ client ทาง socket `client.cfd` ตามขั้นตอนการทำงานของ WWW proxy ที่เคยกล่าวมาแล้ว

5.3 การนำข้อมูล WWW ที่รับ/ส่งผ่านโปรแกรม WWW proxy ไปประมวลผล

หลังจากที่โปรแกรม WWW proxy ได้รับ HTTP request จาก client และส่งต่อ HTTP request ไปยัง WWW server แล้ว เมื่อ WWW server ส่ง HTTP response ตอบกลับการร้องขอที่ส่งไป โปรแกรม WWW proxy จะนำเอา HTTP response พร้อมข้อมูลทรัพยากรที่ client ร้องขอนั้นไปประมวลผลเพื่อเปลี่ยนแปลงแก้ไขข้อมูลหรือกรองข้อมูลก่อนที่จะส่งให้กับ client ที่ร้องขอ

โดย การนำข้อมูลทรัพยากรที่ส่งกลับมาไปประมวลผลนั้น จะอาศัยการสร้าง application program ที่ทำงานเฉพาะอย่างขึ้น ในลักษณะของ plugin program สำหรับโปรแกรม WWW proxy เพื่อให้สามารถเลือกใช้และปรับเปลี่ยนรูปแบบของการประมวลผลข้อมูลได้ตามต้องการ โดยไม่จำเป็นต้องมีการแก้ไขโปรแกรม WWW proxy แต่อย่างใด

ทั้งนี้ ในการส่งข้อมูล WWW จากโปรแกรม WWW proxy ไปยัง application program ที่จะใช้ในการประมวลผลจะกระทำโดยอาศัยหลักการของ DLL (Dynamic Linked Library) โดยจะทำการสร้าง application program ในรูปของ DLL file ซึ่งจะมีการกำหนดฟังก์ชันเริ่มต้นในการทำงานเป็นรูปแบบเดียวกันทั้งหมด เพื่อให้โปรแกรม WWW proxy สามารถเรียกใช้งาน application DLL ต่างๆได้

รูปแบบของฟังก์ชันการทำงานหลักของ application program ได้กำหนดไว้ดังนี้

```
extern "C" __declspec(dllexport) void* ProxyPlugin(
      void *input, long len, long *new_len);
```

โดยจะรับค่าข้อมูลดังนี้

void * *input* เป็นข้อมูล (เอกสาร html หรือข้อมูลอื่นๆ) ที่ส่งมาจากโปรแกรม WWW proxy เพื่อให้ application program นำไปประมวลผล

long *len* เป็นค่าความยาวของข้อมูลที่ส่งมาประมวลผล โดยเป็นตัวเลข ชนิด long

long *`new_len` เป็นค่าความยาวของข้อมูลใหม่ ซึ่งเป็นการคืนค่ากลับไปยังโปรแกรม web proxy ที่ทำการส่งค่าแบบ pass by reference หลังจากได้มีการประมวลผลและแก้ไขแล้วโดยตัว application program แล้ว

ฟังก์ชันนี้จะต้องทำการคืนกลับค่า void * ซึ่งเป็นค่าของข้อมูล (`input`) ที่ได้มีการประมวลผลและแก้ไขไป โดย application program แล้ว

5.3.1 การเรียกใช้งานฟังก์ชันการทำงานของ application program

การเรียกใช้งาน function จาก DLL นั้น สามารถทำได้ใน 2 ลักษณะ คือ Implicit Linkage และ Explicit Linkage โดยที่ทั้งสองวิธีนั้นมีความแตกต่างกันคือ ในการใช้งานแบบ Implicit Linkage นั้น DLL จะถูก load ทันทีเมื่อเริ่มมีการทำงานของโปรแกรม นั่นคือ จะต้องมีการระบุถึงไฟล์ DLL ที่ต้องการใช้งานไว้ล่วงหน้าโดยระบุใน source code ของโปรแกรมที่เรียกใช้ไว้เลย ทำให้ไม่สามารถเปลี่ยนแปลง DLL ที่จะเรียกใช้ได้ในการทำงานของโปรแกรมดำเนินอยู่

แต่ในการเรียกใช้งานแบบ Explicit linkage นั้น ไฟล์ DLL จะถูก load ในขณะที่โปรแกรมที่เรียกใช้งานได้ทำงานอยู่ และมีการเรียกใช้ function การทำงานของ DLL นั้นๆ ดังนั้นจึงสามารถที่จะเลือกใช้ไฟล์ DLL ที่แตกต่างกันได้ในการเรียกใช้งานแต่ละครั้ง

โปรแกรม WWW proxy จะทำการเรียกใช้งาน application program ในแบบ Explicit Linkage เพื่อให้ผู้ใช้สามารถเลือกใช้และเปลี่ยน application ได้ตามต้องการในขณะใช้งานโปรแกรม โดยมีขั้นตอนต่อไปนี้

1. ทำการโหลดไฟล์ application DLL ที่ต้องการโดยฟังก์ชัน `LoadLibrary()`
2. สร้าง pointer ไปยังฟังก์ชันของ DLL ที่ต้องการเรียกใช้โดยฟังก์ชัน `GetProcAddress()`
3. เรียกใช้ฟังก์ชันผ่านทาง pointer ในข้อ 2

5.3.2 การโหลดไฟล์ DLL เพื่อใช้งานด้วยฟังก์ชัน LoadLibrary()

ฟังก์ชัน `LoadLibrary()` ทำการโหลด module ที่ต้องการใช้งานเข้ามาไว้ในเนื้อที่หน่วยความจำของโปรแกรมที่เรียกใช้ โดยมีรูปแบบดังนี้

```
HINSTANCE LoadLibrary( LPCTSTR lpLibFileName );
```

`lpLibFileName` เป็น address ของ module ที่ต้องการโหลดเพื่อใช้งาน (กรณีเป็นไฟล์ DLL *`lpLibFileName`* คือ path และชื่อของไฟล์ DLL นั้น)

ฟังก์ชัน `LoadLibrary()` จะคืนกลับค่า HINSTANCE ซึ่งเป็น Instance Handler สำหรับใช้ในการอ้างอิงถึง module ที่โหลดเข้ามาใช้งาน หรือค่า NULL หากเกิดข้อผิดพลาด และสามารถค้นหารายละเอียดของข้อผิดพลาดนั้นได้ด้วยฟังก์ชัน `GetLastError()`

5.3.3 การเรียกใช้งานฟังก์ชันของ DLL ด้วยฟังก์ชัน GetProcAddress()

ในการเรียกใช้งานฟังก์ชันของไฟล์ DLL ที่ได้ถูกโหลดไว้ในหน่วยความจำ ทำได้โดยการสร้าง pointer ของฟังก์ชันตามรูปแบบของฟังก์ชันที่ต้องการใช้งานนั้น และกำหนดให้ pointer ชี้ไปยัง address ของฟังก์ชันที่ต้องการเรียกใช้ โดยฟังก์ชัน `GetProcAddress()`

```
FARPROC GetProcAddress
(
      HMODULE hModule,
      LPCSTR lpProcName
);
```

hModule เป็น Instance Handler สำหรับไฟล์ DLL ที่ได้จากการคืนค่ากลับของฟังก์ชัน LoadLibrary()

lpProcName เป็นชื่อของฟังก์ชันของไฟล์ DLL ที่ต้องการเรียกใช้งาน

5.3.4 ตัวอย่างการเรียกใช้งาน application program โดยโปรแกรม WWW proxy

```
1.    typedef void* (PLUGIN)(void *, long, long *);
2.    PLUGIN * pFunction;
3.    HINSTANCE hInstance;

4.          ……

5.          VERIFY(hInstance = ::LoadLibrary(plPath));
6.          VERIFY(pFunction = (PLUGIN*)::GetProcAddress
                  (hInstance, "ProxyPlugin"));
7.          ……

8.          all_reply = (char*)(pFunction)(all_reply, n, &n);

9.          ……
```

รูปที่ 5.11 ตัวอย่างการเรียกใช้งาน application program โดยโปรแกรม WWW proxy

บรรทัดที่ (1) – (2) ทำการสร้าง pointer `pFunction` สำหรับฟังก์ชันชนิด `void* (PLUGIN) (void *, long, long *)`

ทำการโหลดไฟล์ DLL (application program) ที่ระบุในตัวแปร plPath โดยฟังก์ชัน `LoadLibrary()` และเก็บค่าจอง Instance Handler ของ DLL นี้ ในตัวแปร HINSTANCE `hInstance` (บรรทัดที่ 5)

กำหนดให้ pointer `pFunction` ชี้ไปยังฟังก์ชันชื่อ `ProxyPlugin` ของไฟล์ DLL `hInstance` (บรรทัดที่ 6)

จากนั้นโปรแกรม WWW proxy สามารถเรียกใช้งานฟังก์ชัน `ProxyPlugin` ของ application program ที่ถูกโหลดไว้ในหน่วยความจำ โดยผ่านทาง pointer `pFunction` (บรรทัดที่ 8)

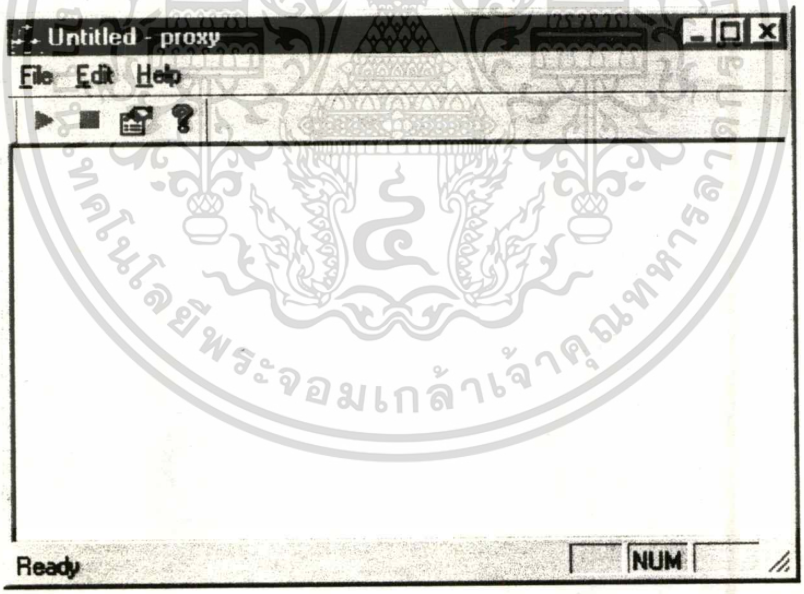
บทที่ 6
สรุปผลการศึกษา

6.1 โปรแกรม WWW proxy และการใช้งาน

จากรูป 6.1 แสดงลักษณะของการติดต่อผู้ใช้ของโปรแกรม web proxy ที่พัฒนาขึ้นในครั้งนี้ โดยได้ทำการออกแบบให้มีการใช้งานแบบ Graphic User Interface ซึ่งมีความสะดวกในการใช้งาน โดยผู้ใช้สามารถที่จะเริ่มใช้งาน web proxy หรือ ยกเลิกการใช้งานได้จากคำสั่ง Start

, Stop


บน menu File หรือโดยปุ่มคำสั่ง Start , Stop บน Toolbar

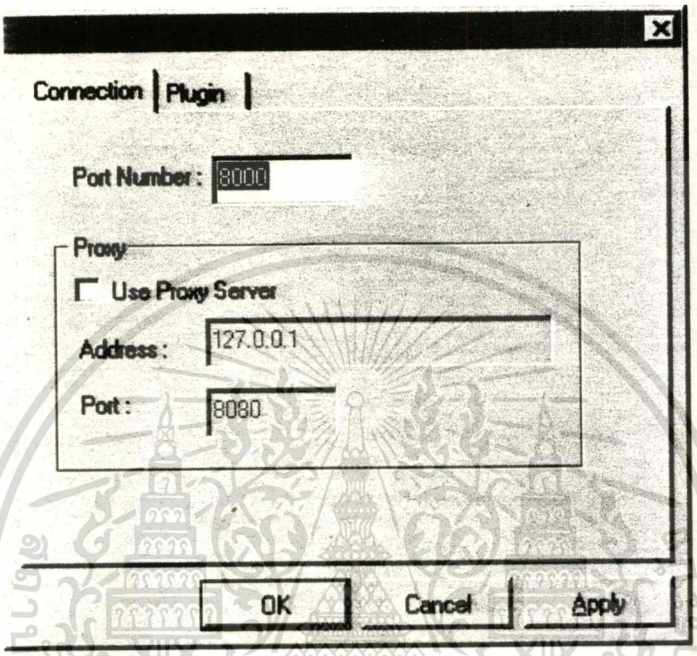


รูปที่ 6.1 แสดงโปรแกรม wen proxy ที่ได้ทำการพัฒนา

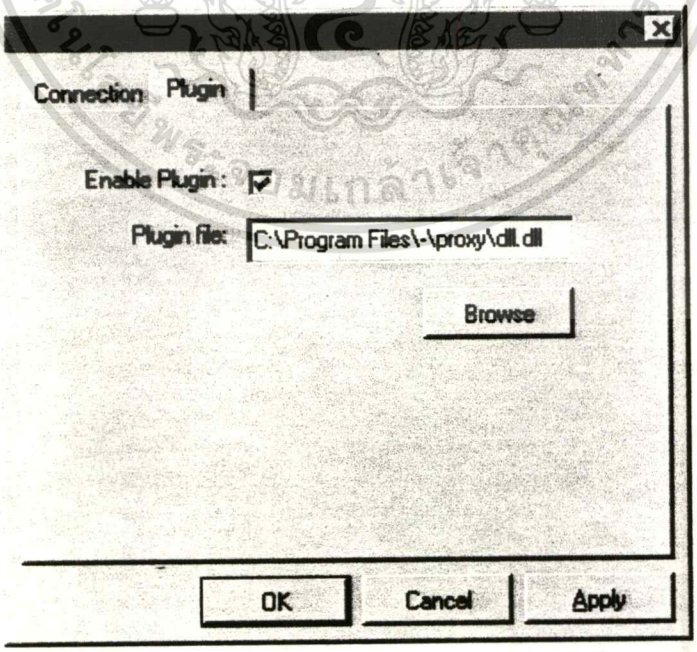
การกำหนดค่าคุณสมบัติ (property) ต่างๆของโปรแกรม สามารถทำได้โดยเลือกคำสั่ง Property บน menu Edit หรือโดยปุ่มคำสั่ง property บน Toolbar ซึ่งจะปรากฏหน้าต่าง property ขึ้นดังรูป

โดยที่ผู้ใช้สามารถกำหนดค่าของ หมายเลข port ที่จะเปิดให้บริการ, และยังสามารถกำหนดการเชื่อมต่อกับเครือขาย Internet โดยอาศัย Proxy หรือ Gateway อื่นๆเป็นตัวกลางในการเชื่อมต่อ

อีกทอดหนึ่งได้ โดยสามารถกำหนดหมายเลข IP Address และหมายเลข port ของ proxy หรือ gateway นั้นๆ ได้ ในแท็ป Connection

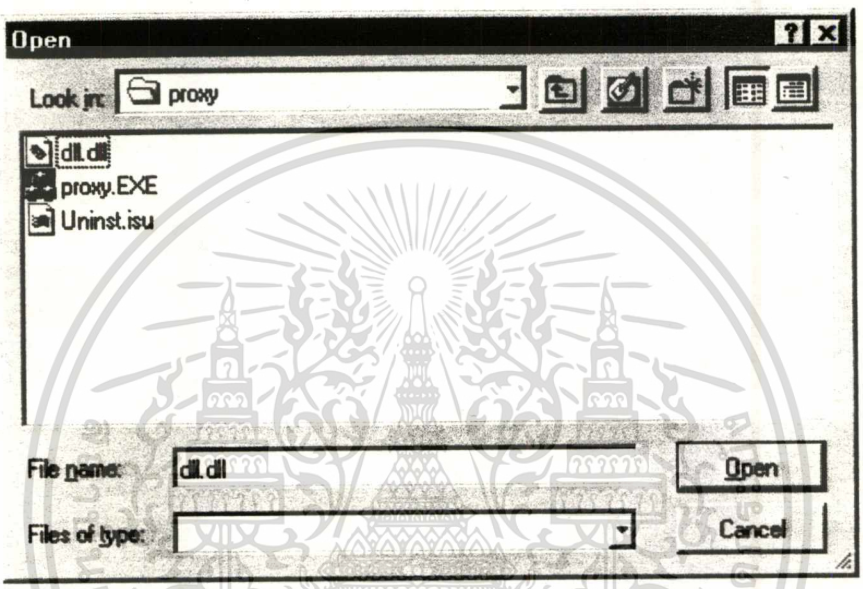


รูปที่ 6.2 แสดง Property connection



รูปที่ 6.3 แสดงProperty Plugin

นอกจากนี้ ผู้ใช้สามารถกำหนดการใช้งาน application อื่นที่ต้องการให้ทำการประมวลผลหน้าข้อมูลที่รับ/ส่งผ่านโปรแกรม web proxy ว่าให้มีการเปิดใช้งานหรือไม่ได้ และสามารถเลือกใช้ application program ที่ต้องการในการประมวลผลนั้นได้โดยระบุชื่อ file DLL ที่ช่อง Plugin file หรือเลือกปุ่ม Browse



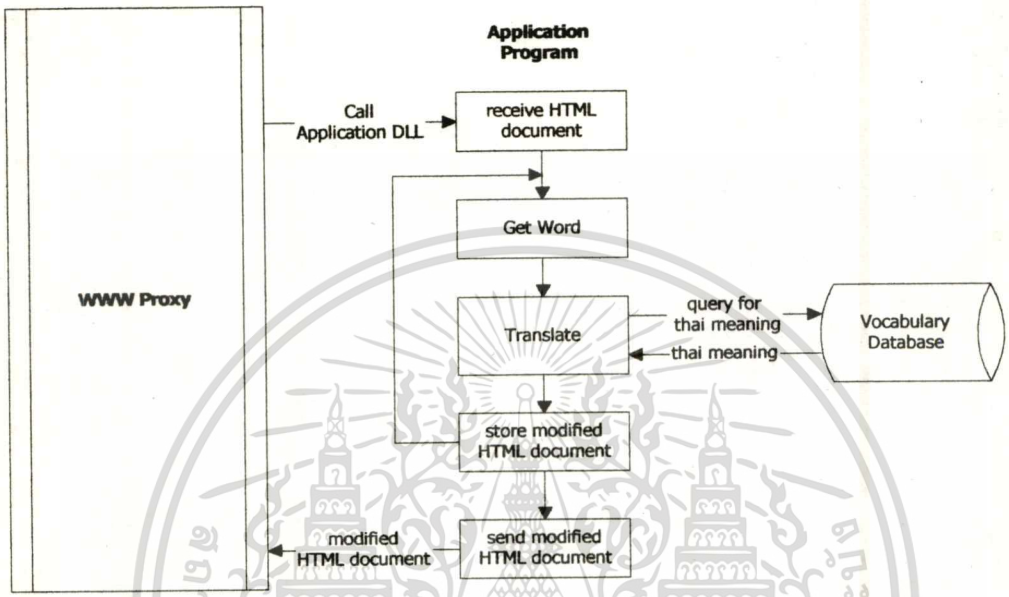
รูปที่ 6.4 แสดงหน้าต่าง open สำหรับเลือกใช้ file application DLL ในการประมวลผลข้อมูล

6.2 Application program ตัวอย่าง : การแปลเอกสาร html ภาษาอังกฤษเป็นภาษาไทย

ในโครงการนี้ ได้มีการพัฒนา Application program ตัวอย่างขึ้นเพื่อใช้ในการประมวลผลข้อมูลเอกสารที่ส่งมาจาก WWW proxy โดยโปรแกรมัวอย่างนั้นทำหน้าที่ในการแปลความหมายของคำศัพท์ภาษาอังกฤษเป็นคำแปลภาษาไทย โดยจะประกอบด้วยไฟล์ “dictionary.dll” ซึ่งเป็นไฟล์ DLL ของ Application program ดังกล่าว และไฟล์ “dictionary.mdb” ซึ่งเป็นไฟล์ฐานข้อมูลสำหรับจัดเก็บคำศัพท์ภาษาอังกฤษและคำแปลภาษาไทย

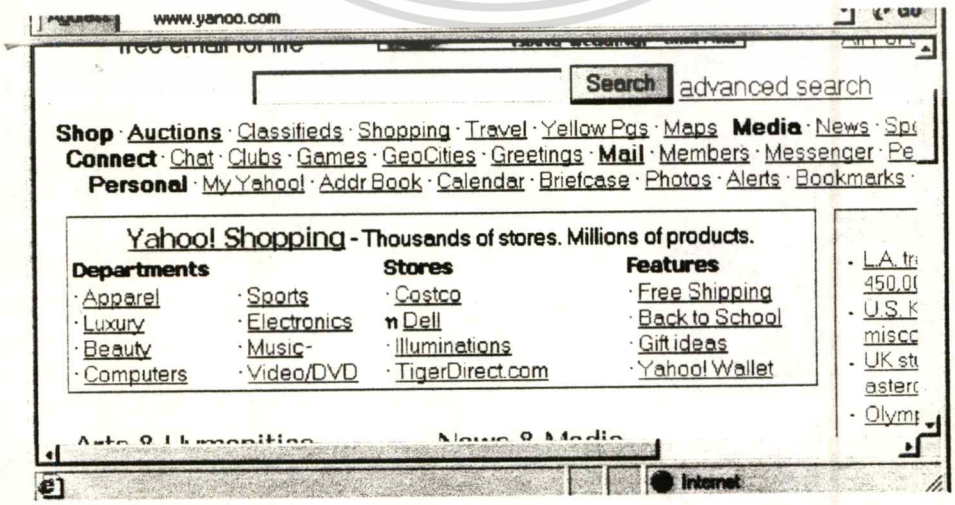
ในการทำงานนั้น หลังจากโปรแกรม WWW proxy ได้ส่งข้อมูลเอกสารมายังโปรแกรม dictionary แล้ว โปรแกรม dictionary จะทำการแยกคำศัพท์ทีละคำออกมาจากเอกสาร HTML และทำการค้นหาค้นหาคำแปลภาษาไทยสำหรับคำศัพท์ภาษาอังกฤษนั้นๆ โดยการค้นหาในฐานข้อมูลของคำศัพท์ที่เตรียมไว้ จากนั้นจึงทำการเปลี่ยนแปลงข้อความในเอกสารตรงตำแหน่งที่เป็นคำศัพท์นั้นโดยแทนที่ด้วยคำแปลภาษาไทยแทน แต่หากคำศัพท์คำใดไม่พบว่ามีคำแปลภาษาไทยบันทึกไว้ในฐานข้อมูล ก็จะคงคำศัพท์นั้นไว้อย่างเดิม ทั้งนี้โปรแกรม dictionary ที่พัฒนาขึ้นนี้ ทำการแปล

ความหมายของศัพท์ภาษาอังกฤษแบบคำต่อคำ โดยจะไม่ครอบคลุมถึงความหมายของทั้งประโยค, การตัดคำ และการจัดรูปประโยคภาษาไทยแต่อย่างใด



รูปที่ 6.5 แสดงขั้นตอนการทำงานของโปรแกรม dictionary

รูปที่ 6.6 และรูปที่ 6.7 แสดงหน้าเอกสาร HTML ตัวอย่าง โดยเป็นหน้าเอกสารต้นฉบับที่ไม่ได้ผ่านการประมวลผลเพื่อเปลี่ยนแปลงข้อมูลโดยโปรแกรม dictionary สำหรับ WWW proxy และหน้าเอกสาร HTML ตัวอย่างที่ได้ถูกเปลี่ยนแปลงแก้ไขแล้วโดยโปรแกรม dictionary สำหรับ WWW proxy ที่พัฒนาในครั้งนี้ ตามลำดับ



รูปที่ 6.6 แสดงหน้าเอกสาร HTML (เอกสารต้นฉบับ)

รูปที่ 6.7 แสดงหน้าเอกสาร HTML ที่ได้ถูกเปลี่ยนแปลงแก้ไขแล้วโดยโปรแกรม dictionary

6.3 สรุปผลการศึกษา

จากการศึกษาและพัฒนาโปรแกรม WWW proxy สามารถทำการพัฒนาโปรแกรม WWW proxy ต้นแบบที่ทำการส่งข้อมูลติดต่อกับ application program เพื่อนำข้อมูลไปประมวลผลก่อนการนำไปแสดงผลทาง WWW browser โดยอาศัยลักษณะการทำงานของโปรแกรม WWW proxy เอง ที่ทำหน้าที่เป็นตัวกลางในการรับ/ส่งข้อมูลระหว่าง web server ผู้ให้บริการกับ web browser ผู้ใช้บริการทรัพยากรต่างๆ ทำให้สามารถนำเอาข้อมูลทรัพยากร (เอกสาร html และข้อมูลอื่นๆ) เหล่านี้มาทำการประมวลผลก่อนที่จะส่งต่อให้กับ web browse เพื่อแสดงผลได้

โดยในส่วนของการประมวลผลข้อมูลนั้น จะอาศัยการทำงานของ application program ที่ถูกพัฒนาขึ้นในรูปแบบของ DLL (Dynamic Link Library) เพื่อให้โปรแกรม WWW proxy เรียกใช้งานได้ตามแต่ความเหมาะสมของการทำงานของแต่ละ application program ได้อย่างอิสระ

และในส่วนของโปรแกรม WWW proxy ที่พัฒนาขึ้นนั้น สนับสนุนการรับ/ส่งข้อมูลเอกสาร html มาตรฐาน, ไฟล์รูปภาพและไฟล์ข้อมูลมาตรฐานอื่นๆ รวมถึงสนับสนุนการติดต่อสื่อสารผ่านทาง proxy server หรือ gateway ในการเชื่อมต่อเครือข่ายอินเตอร์เน็ตอีกด้วย ทั้งนี้ยังสามารถพัฒนาเพื่อเติมในส่วนของคุณสมบัติการ caching ข้อมูลที่มีการรับ/ส่ง รวมไปถึงการสนับสนุนการทำงานของ cookies ด้วย

บรรณานุกรม

Chapman, D.B. and Zwicky E.D. 1998. **Building Internet Firewalls.** New York : O'Reilly & Associates.

Dingle, A. **Web Cache Coherence.** Available : http://sun3.ms.mff.cuni.cz/~dingle/webcoherence.html

Fielding, R., et al. 1997. **RFC 2068: Hypertext Transfer Protocol – HTTP/1.1.**

Glass, G and Rabinovich, M. **Web Proxy Caching: The Devil is in the Details.** Available : http://www.cs.wisc.edu/~cao/wisp98/html-versions/anja/proxim-wisp/

Irvine, U.C. and Frystyk, H. 1996. **RFC 1945: Hypertext Transfer Protocol – HTTP /1.0.**

Kruglinski, D.J. et.al.1998. **Programming Microsoft VisualC++.** 5th ed. Washington : Microsoft Press.

Stevens, W.R.1998. **UNIX Network Programming.** Vol. 1 2nd ed. New Jersey : Prentice Hall.

Wessels, D. **Information Resource Caching FAQ.** Available : http://ircache.nlanr.net/Cache/FAQ/ircache-faq.html

Yeager, D.B. and McGrath, R.E. 1996. **Web Server Technology.** : Morgan Kaufmann.

ภาคผนวก

source code โปรแกรม dictionary.dll

โปรแกรม dictionary ถูกพัฒนาในรูปแบบของ Dynamic Linked Library ด้วยภาษา Visual C++ โดยประกอบส่วนส่วนประกอบต่างๆดังนี้

1. ไฟล์ dictionary.h

```
// dictionary.h : main header file for the DICTIONARY DLL

#if !defined(AFX_DICTIONARY_H__E02F7006_9465_11D4_9B7D_86A4DD38862D__
INCLUDED_)
#define AFX_DICTIONARY_H__E02F7006_9465_11D4_9B7D_86A4DD38862D__INCLU
DED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
      #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"        // main symbols

extern "C" __declspec(dllexport) void* ProxyPlugin(void *input, long
len, long *new_len);

void getTag(int *begin, CString sinput, char **tag);
void getWord(int *begin, CString sinput, char **word);
int translate(char*, char**, CDaoDatabase*);

class CDictionaryApp : public CWinApp
{
public:
      CDictionaryApp();
// Overrides
      //{{AFX_VIRTUAL(CDictionaryApp)
      public:
      virtual BOOL InitInstance();
      virtual int ExitInstance();
      //}}AFX_VIRTUAL
      //{{AFX_MSG(CDictionaryApp)
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
#endif
```

2. ไฟล์ dictionary.cpp

```
// dictionary.cpp : Defines the initialization routines for the DLL.

#include "stdafx.h"
#include "dictionary.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CDictionaryApp

BEGIN_MESSAGE_MAP(CDictionaryApp, CWinApp)
      //{{AFX_MSG_MAP(CDictionaryApp)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// CDictionaryApp construction

CDictionaryApp::CDictionaryApp()
{
}

// The one and only CDictionaryApp object

CDictionaryApp theApp;

void getTag(int *begin, CString sinput, char **tag)
{
 int end, len;

      end = sinput.Find(">", (*begin));
      if (end == -1) {
            end = sinput.GetLength() -1;
      }

      len = end - (*begin) +1;
      **tag = strdup(sinput.Mid(*begin, len));
      *begin += len;
}

void getWord(int *begin, CString sinput, char **word)
{
 int end, len;

      end = (*begin) + 1;
      while ((sinput[end]>=65 && sinput[end]<=90) ||
                  (sinput[end]>=97 && sinput[end]<=122)) {
            end++;
      }
      len = end - (*begin);

      *word = strdup(sinput.Mid(*begin, len));
      *begin += len;
}
```

```
int translate(char* eng, char** thai, CDaoDatabase *db)
{
 CDaoRecordset *rs;
 COleVariant out;
 char buf[2000];

      sprintf(buf, "SELECT thai FROM EngtoThai where eng='%s'", eng);
      try {
            rs = new CDaoRecordset(db);
            rs->Open(dbOpenDynaset, buf, dbReadOnly);
            rs->GetFieldValue("thai", out);
            *thai = strdup(out.pcVal);
            *thai = strsav(*thai, "\0");
            rs->Close();
      }
      catch (CDaoException *e) {
            *thai = eng;
            e->Delete();
            return 0;
      }
      return strlen(*thai)-strlen(eng);
}

void* ProxyPlugin(void* input, long len, long *new_len)
{
 int ptr, Cend, clen;
 CDaoDatabase *db;
 char *nocache = "Pragma: no-cache\n";
 char *str, *thai, cur;

      CString sinput((char*)input);

      str = "\r\n\r\n";
      ptr = sinput.Find("Content-Type: text/html", 0);
      if (ptr     == -1) {
            //input is not a text/html, do nothing
            (*new_len) = len;
            return input;
      }

      try {
                  AfxDaoInit();
                  db = new CDaoDatabase;
                  db->Open("dictionary.mdb", TRUE, FALSE, "");
      }
      catch (CDaoException *e) {
            AfxMessageBox(e->m_pErrorInfo->m_strDescription);
            e->Delete();
            AfxDaoTerm();
            (*new_len) = len;
            return input;
      }

      ptr = sinput.Find(str, 0);
      if (ptr == -1) {
      //find HTTP Entity
            (*new_len) = len;
            return input;
```

```
        }
        ptr += 4;

        CString soutput = sinput.Left(ptr);

        int lenChange =0;
        while (ptr < sinput.GetLength()) {
              cur = sinput.GetAt(ptr);
              if (cur == '<') {
                    getTag(&ptr, sinput, &str);
                    soutput += str;
              }
              else if ((cur>=65 && cur<=90) ||
                          (cur>=97 && cur<=122)) {
                    getWord(&ptr, sinput, &str);
                    lenChange += translate(str, &thai, db);

                    soutput += thai;
              }
              else {
                    soutput += strdup(sinput.Mid(ptr, 1));
                    ptr++;
              }
        }

        // update "Content-Length" header

        ptr = soutput.Find("Content-Length", 0);
        Cend = soutput.Find("\n", ptr);

        str = strdup(soutput.Mid(ptr+16, Cend - (ptr+16) +1));

        sscanf(str,"%d\n", &clen);
        clen += lenChange;

        soutput.Delete(ptr, Cend - ptr + 1);

        CString Stemp;
        Stemp.Format("Content-Length: %d\n", clen);
        soutput.Insert(ptr, Stemp);
        soutput.Insert(ptr, nocache);

        AfxDaoTerm();

        str = strdup(soutput.GetBuffer(sinput.GetLength()));
        (*new_len) = strlen(str);
        return str;
}

BOOL CDictionaryApp::InitInstance()
{
        return CWinApp::InitInstance();
}

int CDictionaryApp::ExitInstance()
{
        return CWinApp::ExitInstance();
}
```

3. ไฟล์ stdafx.h

```
// stdafx.h : include file for standard system include files

#if !defined(AFX_STDAFX_H__E02F7008_9465_11D4_9B7D_86A4DD38862D__INCL
UDED_)
#define AFX_STDAFX_H__E02F7008_9465_11D4_9B7D_86A4DD38862D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN    /* Exclude rarely-used stuff from Windows
headers */

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions

#ifndef _AFX_NO_OLE_SUPPORT
#include <afxole.h>         // MFC OLE classes
#include <afxodlgs.h>       // MFC OLE dialog classes
#include <afxdisp.h>        // MFC Automation classes
#endif // _AFX_NO_OLE_SUPPORT


#ifndef _AFX_NO_DB_SUPPORT
#include <afxdb.h>                  // MFC ODBC database classes
#endif // _AFX_NO_DB_SUPPORT

#ifndef _AFX_NO_DAO_SUPPORT
#include <afxdao.h>                 // MFC DAO database classes
#endif // _AFX_NO_DAO_SUPPORT

#include <afxdtctl.h>           // MFC support for Internet Explorer 4
Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                 // MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}

#endif
```

4. ไฟล์ stdafx.cpp

```
// stdafx.cpp : source file that includes just the standard includes

#include "stdafx.h"
```

ประวัติผู้เขียน

ชื่อ-นามสกุล	นายอภิชาติ ศรีเพียร
วัน/เดือน/ปีเกิด	18 ธันวาคม 2520
ที่อยู่	150/29 ซ.วัดบรมนิวาส ถ.พระราม 1 แขวงรองเมือง เขตปทุมวัน กทม. 10330
ประวัติการศึกษา	วท.บ. วิทยาการคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย