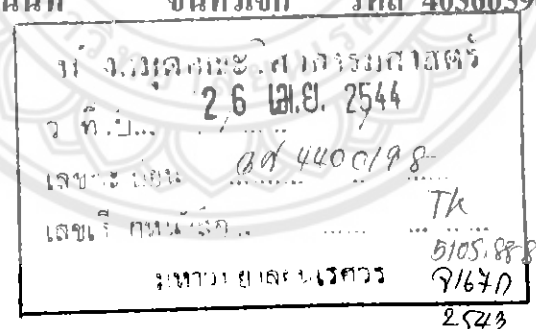
การพัฒนาโปรแกรมสำหรับเครื่องคอมพิวเตอร์แม่ข่าย เพื่อถ่ายโอนข้อมูลแบบเอชทีทีพี (HTTP)

A Development of the Server Program for HTTP Data Transferring

นางสาวจริยา มุ่งคีมกลาง รหัส 40360216

นางสาวจันทร์นิภา จันทร์แจ้ง รหัส 40360257

นายอานนท์ จันทร์เจ๊ก รหัส 40360596



โครงงานนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร

ปีการศึกษา 2543

ใบรับรองโครงงานวิจัย

หัวข้อโครงงาน,	:	การพัฒนาโปรแกรมสำหรับเครื่องคอมพิวเตอร์แม่ข่าย เพื่อถ่ายโอนข้อมูลแบบเอชทีทีพี (HTTP)
Performance	:	A Development of the Server Program for HTTP Data Transferring
ผู้ดำเนินโครงงาน	:	นางสาวจริยา มุ่งคีมกลาง รหัส 40360216
		นางสาวจันทร์นิภา จันทร์แจ้ง รหัส 40360257
		นายอานนท์ จันทร์เจ๊ก รหัส 40360596
อาจารย์ที่ปรึกษา	:	อาจารย์อธิศ ปทุมวรรณ
อาจารย์ที่ปรึกษาร่วม	:	อาจารย์แคทรียา อัดสูงเนิน
สาขา	:	วิศวกรรมคอมพิวเตอร์
ภาควิชา	:	วิศวกรรมไฟฟ้าและคอมพิวเตอร์

.....

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร อนุมัติให้โครงงานฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร วิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์

คณะกรรมการสอบโครงงานวิจัย

.....ประธานกรรมการ
(อาจารย์อธิศ ปทุมวรรณ)

.....กรรมการ
(อาจารย์วัชร์วีร์ พืชพันธ์)

.....กรรมการ
(อาจารย์ประทีป ตรีรณโอภาส)

หัวข้อโครงงาน	:	การพัฒนาโปรแกรมสำหรับเครื่องคอมพิวเตอร์แม่ข่าย เพื่อถ่ายโอนข้อมูลแบบเอชทีทีพี (HTTP)
ผู้ดำเนินโครงงาน	:	นางสาวจริยา มุ่งคีมกลาง รหัส 40360216
		นางสาวจันทร์นิภา จันทร์แจ้ง รหัส 40360257
		นายอานนท์ จันทร์เจ๊ก รหัส 40360596
อาจารย์ที่ปรึกษา	:	อาจารย์อธิศ ปทุมวรรณ
อาจารย์ที่ปรึกษาร่วม	:	อาจารย์แคทรียา อัดสูงเนิน
สาขา	:	วิศวกรรมคอมพิวเตอร์
ภาควิชา	:	วิศวกรรมไฟฟ้าและคอมพิวเตอร์
ปีการศึกษา	:	2543

.....

บทคัดย่อ

โครงงานนี้เป็นการศึกษาและพัฒนาโปรแกรมสำหรับใช้รันบนเครื่องคอมพิวเตอร์แม่ข่าย (Server) เพื่อการถ่ายโอนข้อมูลโดยใช้โปรโตคอลเอชทีทีพี (HTTP Protocol) หรือเรียกอีกอย่างหนึ่งว่าโปรแกรมเว็บเซิร์ฟเวอร์ (Web Server Program) ซึ่งเป็นโปรแกรมที่ให้บริการเว็บเพจ (Web Page) บนระบบเครือข่ายอินเตอร์เน็ต โดยโปรแกรมสามารถเข้าถึงจากระยะไกลผ่านเครื่องคอมพิวเตอร์ลูกข่าย (Client) ด้วยวิธีการร้องขอแบบต่าง ๆ เช่น การร้องขอเว็บเพจจากเว็บเบราเซอร์ (Web Browser)

การพัฒนาโปรแกรมนี้ทดสอบและรันบนระบบปฏิบัติการไมโครซอฟท์วินโดวส์ 98 ใช้ภาษาวิชวลซีพลัสพลัส เวอร์ชั่น 6.0 (Visual C++ version 6.0) ในการพัฒนา และอ้างอิงการทำงานตามวินโดวส์ซ็อกเก็ตเอพีไอ 2.0 (Windows Socket API 2.0) และมาตรฐานอาร์เอฟซี 1945 (RFC 1945)

ผลที่ได้จากการทำโครงงานนี้ คือ ได้โปรแกรมที่สามารถให้บริการเว็บเพจบนระบบเครือข่าย สามารถรองรับการทำงานด้วยวิธีการร้องขอแบบ GET และ HEAD ตามมาตรฐานเอชทีทีพี เวอร์ชั่น 1.0 (HTTP version 1.0) ซึ่งเป็นมาตรฐานที่เครื่องคอมพิวเตอร์แม่ข่ายใช้กันอยู่ในปัจจุบัน โดยโปรแกรมนี้สามารถให้บริการการถ่ายโอนข้อมูลประเภทอักขระ รูปภาพ และข้อมูลแบบมัลติมีเดียได้

Project Title	:	A Development of the Server Program for HTTP Data Transferring
Name	:	Miss Jariya Moongkheemklang ID. 40360216
		Miss Channipa Chanchang ID. 40360257
		Mr. Anondh Chanchek ID. 403600596
Project Advisor	:	Mr. Atit Pathumwan
Co- Project Advisor	:	Miss Cattareeya Adsoongneon
Field of Study	:	Computer Engineering
Department	:	Electrical and Computer Engineering
Academic Year	:	2000

Abstract

This project is studied and developed a program for the server to transfer data with HTTP protocol or we called Web Server Program . This program can be serviced web page on internet network and can be accessed remotely from client by request method such as the request from web browser .

A Development this program tests and runs on Microsoft Windows 98 Operating System . This program is developed using Visual C++ version 6.0 language , refers to Windows Socket API 2.0 and RFC 1945 .

The result of this project is the program that can service web page by method GET and HEAD , refers to HTTP version 1.0 which is the standard for server in this present . This program can transfer text data , picture and multimedia .

กิตติกรรมประกาศ

โครงงานนี้สำเร็จไปได้ด้วยความช่วยเหลืออย่างดียิ่งจากอาจารย์อธิศ ปทุมวรรณ อาจารย์ที่ปรึกษาโครงงาน และอาจารย์แคทรียา อัดสูงเนิน อาจารย์ที่ปรึกษาร่วม ท่านได้สละเวลา ความคิด ประสบการณ์ และคำปรึกษาแนะนำ ในการทำโครงงานจนกระทั่งแล้วเสร็จ ทำให้ผู้คณะผู้จัดทำได้รับประสบการณ์ทำงานอันมีค่ายิ่ง

ขอขอบคุณอย่างสูงต่อ อาจารย์สุเมธ อังคะศิริกุล ที่ให้การสนับสนุนในการสืบค้นข้อมูลเกี่ยวกับโครงงานการพัฒนาโปรแกรม

นางสาวจริยา มุ่งดีมกลาง
นางสาวจันทร์นิภา จันทร์แจ้ง
นายอานนท์ จันทร์เจ๊ก

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

หน้า

สารบัญรูปภาพ

หน้า

สารบัญรูปภาพ (ต่อ)

หน้า

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของโครงงาน

ในปัจจุบันนี้ เทคโนโลยีด้านระบบเครือข่ายอินเตอร์เน็ตได้ถูกพัฒนาไปอย่างรวดเร็ว และได้เข้ามามีบทบาทกับชีวิตประจำวันของเรามากขึ้นตามลำดับ เนื่องจากสังคมปัจจุบันเป็นยุคของสังคมข่าวสารและสารสนเทศ ความถูกต้อง ความรวดเร็วและความทันสมัยของข้อมูลจึงเป็นสิ่งที่สำคัญมาก ระบบเครือข่ายอินเตอร์เน็ตเป็นแหล่งข้อมูลสารสนเทศขนาดใหญ่ ที่สามารถให้บริการข้อมูลที่เป็นภาพและเสียงได้อย่างประสิทธิภาพ จึงทำให้ถูกนำมาใช้ในงานแขนงต่าง ๆ อย่างแพร่หลายนั่นเอง

บนระบบเครือข่ายอินเตอร์เน็ตมีเทคโนโลยีหรือมาตรฐานที่เกี่ยวข้องอยู่มากมาย เช่น มาตรฐาน TCP/IPเทคโนโลยีการทำงานของเครื่องแม่ข่ายแบบ FTP และ แบบ HTTP เป็นต้น นอกจากนั้นบนระบบเครือข่ายอินเตอร์เน็ตยังมีแอปพลิเคชันต่าง ๆ มากมาย ที่เป็นประโยชน์ต่อผู้ใช้งานในระบบโดยแอปพลิเคชันต่างๆ เหล่านี้ก็มีมาตรฐานแตกต่างกันออกไป ตามลักษณะการใช้งานนั่นเอง

ถึงแม้ความก้าวหน้าทางการติดต่อสื่อสารบนระบบเครือข่ายอินเตอร์เน็ตจะเป็นไปอย่างรวดเร็ว แต่ก็ยังพบว่าการพัฒนาเทคโนโลยีบางแขนงในประเทศไทย ยังเป็นไปภายในวงแคบ เช่น การพัฒนาโปรแกรมเพื่อให้บริการหรือโปรแกรมสำหรับการถ่ายโอนข้อมูลในเครื่องคอมพิวเตอร์แม่ข่ายแบบ FTP และแบบ HTTP เป็นต้น

สิ่งที่ผู้จัดทำโครงงานต้องการศึกษาและพัฒนาก็คือ โปรแกรมสำหรับเครื่องคอมพิวเตอร์แม่ข่ายเพื่อถ่ายโอนข้อมูลแบบ HTTP เนื่องจากเป็นเทคโนโลยีที่มีราคาถูกและไม่ขึ้นอยู่กับฮาร์ดแวร์ โดยการพัฒนาโปรแกรมนี้ได้อ้างอิงกับมาตรฐาน RFC 1945 และ HTTP เวอร์ชั่น 1.0 ซึ่งได้รองรับการร้องขอ 3 แบบ คือ

1. แบบ GET คือ การร้องขอให้เซิร์ฟเวอร์ ส่งไฟล์มาให้ หรือร้องขอเพื่อถามเซิร์ฟเวอร์ว่ามีไฟล์ที่ต้องการอยู่ในเซิร์ฟเวอร์หรือไม่

2. แบบ HEAD คือ การร้องขอเพื่อให้เซิร์ฟเวอร์ส่งเฮดเดอร์ของไฟล์มาให้ เพื่อจะตรวจสอบว่าไฟล์นั้น ๆ มีการเปลี่ยนแปลงหรือไม่

3. แบบ POST คือ การร้องขอให้เซิร์ฟเวอร์รับข้อมูลจากไคลเอนต์ หมายความว่า มีไคลเอนต์ต้องการส่งข้อมูลให้กับเซิร์ฟเวอร์

โดยโปรแกรมที่พัฒนาขึ้นนี้ สามารถรองรับ HTML แบบข้อความและภาพกราฟิกแบบธรรมดาได้คือเป้าหมายของการทำโครงการนี้

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาเทคโนโลยีและมาตรฐานต่าง ๆ เหล่านี้
   - มาตรฐานโปรโตคอล TCP/IP
   - มาตรฐานโปรโตคอล IPv6
   - เทคโนโลยี RFC 1945
   - เทคโนโลยี HTTP เวอร์ชั่น 1.0
   - วินโดวส์ซ็อกเก็ต
   - การให้บริการตามคำร้องขอของเครื่องคอมพิวเตอร์ลูกข่าย (client)
   - การพัฒนาโปรแกรมบนระบบปฏิบัติการวินโดวส์
   - การถ่ายโอนข้อมูลระหว่างเครื่องคอมพิวเตอร์แม่ข่ายกับเครื่องคอมพิวเตอร์ลูกข่าย
2. เพื่อศึกษาและพัฒนาโปรแกรมสำหรับเครื่องคอมพิวเตอร์แม่ข่ายเพื่อถ่ายโอนข้อมูลแบบ HTTP
3. เพื่อพัฒนาโปรแกรมให้บริการอ้างอิงตามมาตรฐานRFC1945และ HTTPเวอร์ชั่น1.0
4. เพื่อศึกษาและพัฒนาเครื่องคอมพิวเตอร์ที่ทำงานเป็นเครื่องแม่ข่ายแบบ HTTP

1.3 ขอบข่ายการทำงาน

1. พัฒนาโปรแกรมสำหรับเครื่องแม่ข่ายเพื่อถ่ายโอนข้อมูลแบบ HTTP ภายใต้สภาวะแวดล้อมไมโครซอฟท์วินโดวส์ 98
2. โปรแกรมที่ได้พัฒนาขึ้นอ้างอิงตามมาตรฐาน RFC 1945 และ HTTP เวอร์ชั่น 1.0
3. การถ่ายโอนข้อมูลจากเครื่องคอมพิวเตอร์แม่ข่ายไปสู่เครื่องคอมพิวเตอร์ลูกข่าย

1.4 ขั้นตอนการทำงาน

1. ศึกษาสิ่งต่าง ๆ ต่อไปนี้
   - มาตรฐานโปรโตคอล TCP/IP และ IPv6
   - เทคโนโลยี RFC 1945 และ HTTP เวอร์ชั่น 1.0
   - วินโดวส์ซ็อกเก็ต
   - การให้บริการตามคำร้องขอของเครื่องคอมพิวเตอร์ลูกข่าย
   - การพัฒนาโปรแกรมบนระบบปฏิบัติการวินโดวส์
   - การถ่ายโอนข้อมูลระหว่างเครื่องคอมพิวเตอร์แม่ข่าย กับเครื่องคอมพิวเตอร์ลูกข่าย
2. ออกแบบโปรแกรม อัลกอริทึม โครงสร้างและวิธีการตามคำร้องขอ โดยรวบรวมตามความต้องการที่ได้ศึกษาจากข้อ 1 แล้วนำมาประยุกต์หรือดัดแปลง
3. ทำการพัฒนาโปรแกรม โดยโปรแกรมจะถูกพัฒนาบนเครื่องคอมพิวเตอร์ส่วนบุคคล ซึ่งทำงานภายใต้สภาวะแวดล้อมวินโดวส์ 98 (โดยใช้ภาษาวิชวลซีพลัสพลัสในการพัฒนาโปรแกรม)
4. ทดสอบโปรแกรม
5. ปรับปรุงแก้ไขโปรแกรม
6. สรุปผล

ตารางที่ 1.1 กิจกรรมการดำเนินงาน

กิจกรรม	เดือนปี						
	มี.ค 43	เม.ย 43	พ.ค 43	มิ.ย 43	ก.ค 43	ส.ค 43	ก.ย 43
1. ศึกษาสิ่งต่าง ๆ ที่เกี่ยวข้อง	←	—	→				
2. ออกแบบโปรแกรม อัลกอริทึม โครงสร้าง		←	—	→			
3. ทำการพัฒนาโปรแกรม			←	—	—	→	
4. ทดสอบโปรแกรม					←	—	→
5. ทำการปรับปรุงแก้ไขโปรแกรม						←	— →
6. สรุปผล							← — →

1.5 ผลที่คาดว่าจะได้รับ

1. ได้รับความรู้เกี่ยวกับเทคโนโลยีและมาตรฐานต่าง ๆ บนระบบเครือข่ายอินเตอร์เน็ตที่ได้ทำการศึกษา

2. ได้โปรแกรมสำหรับเครื่องคอมพิวเตอร์แม่ข่ายเพื่อถ่ายโอนข้อมูลแบบ HTTP ตามมาตรฐาน RFC 1945 และ HTTP เวอร์ชั่น 1.0

3. ได้แนวทางและวิธีการพัฒนาโปรแกรมที่เกี่ยวข้องกับระบบเครือข่ายอินเตอร์เน็ต และรูปแบบวิธีการสื่อสารของข้อมูล โดยใช้โปรโตคอล TCP/IP

1.6 งบประมาณที่ต้องใช้ในการดำเนินงาน

1.	ค่าจัดซื้อหนังสือ	550 บาท
2.	ค่าพิมพ์และถ่ายเอกสาร	1,540 บาท
3.	ค่าการ์ดแลนและสายยูทีพี	910 บาท
	รวมเป็นเงินทั้งสิ้น	3,000 บาท

*โดยขออนุมัติถัวเฉลี่ยทุกรายการ

บทที่ 2
ความรู้เบื้องต้นสำหรับการเขียนโปรแกรมบนระบบเครือข่าย

ในบทที่ 2 นี้ เป็นเนื้อหาของหลักการและทฤษฎีที่เกี่ยวข้องในการทำโครงงานนี้ เนื่องจากการสื่อสารข้อมูลบนระบบเครือข่ายต้องใช้มาตรฐานการรับส่งข้อมูลต่าง ๆ จึงมีความจำเป็นที่จะต้องศึกษาและทำความเข้าใจกลไกการทำงานและมาตรฐานต่าง ๆ ให้ถ่องแท้ รวมทั้งยังต้องมีความเข้าใจการพัฒนาโปรแกรมบนระบบเครือข่ายอีกด้วย โดยในบทนี้ประกอบด้วยเนื้อหาหลัก ๆ 3 ส่วนดังนี้ โปรโตคอลทีซีพี/ไอพี (TCP/IP Protocol) และโอเอสไอโมเดล โปรโตคอลเอชทีทีพี วินโดวส์ซ็อกเก็ตและการพัฒนาโปรแกรมบนระบบเครือข่าย

2.1 โปรโตคอลทีซีพี/ไอพี (TCP/IP Protocol) และโอเอสไอโมเดล (OSI Model)

1. โปรโตคอลคืออะไร

โปรโตคอล (Protocol) คือ ระเบียบวิธีที่กำหนดขึ้นสำหรับการสื่อสารข้อมูล ให้สามารถส่งผ่านข้อมูลไปยังปลายทางได้อย่างถูกต้อง ในปัจจุบันโปรโตคอลในการสื่อสารข้อมูลก็มีอยู่หลายโปรโตคอลนอกเหนือจากทีซีพี/ไอพี (TCP/IP) เช่น โปรโตคอลไอพีเอ็กซ์/เอสพีเอ็กซ์ (IPX/SPX), โปรโตคอลเน็ทไบออส (NetBIOS) และโปรโตคอลแอปเปิ้ลทอล์ค (Apple Talk) เป็นต้น

2. โปรโตคอลทีซีพี/ไอพี (TCP/IP)

โปรโตคอลทีซีพี/ไอพี (TCP/IP) เป็นชื่อเรียกของชุดโปรโตคอลที่สำคัญมีการใช้งานกันอย่างแพร่หลายตามการขยายตัวของอินทราเน็ต/อินเตอร์เน็ต ความจริงแล้ว โปรโตคอลทีซีพี/ไอพี (TCP/IP) เป็นกลุ่มโปรโตคอลหลายตัวที่ประกอบกันเป็นชุดให้ใช้งาน โดยมีส่วนประกอบหลักๆ 2 ส่วน คือ

TCP ย่อมาจาก Transmission Control Protocol มีหน้าที่ในการตรวจสอบการรับส่งข้อมูลระหว่างคอมพิวเตอร์ผู้รับและผู้ส่ง ให้ได้รับข้อมูลอย่างถูกต้องครบถ้วน หากข้อมูลสูญหายก็จะแจ้งให้ต้นทางส่งมาใหม่

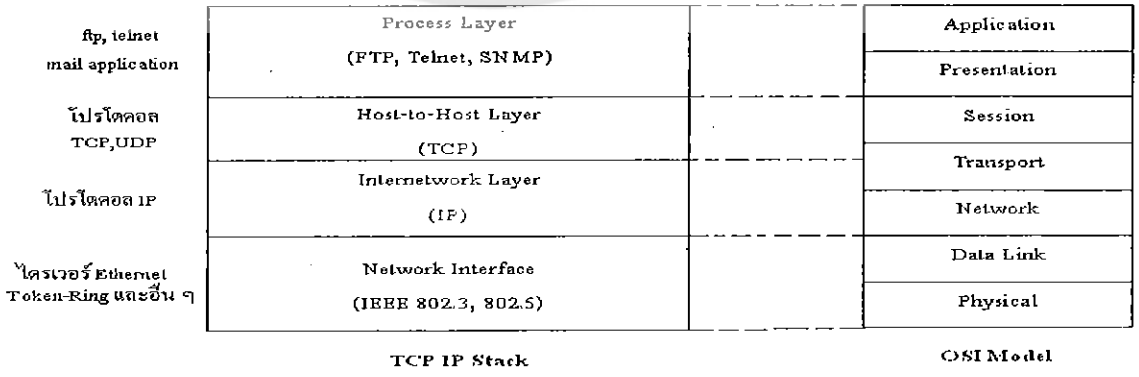
IP ย่อมาจาก Internet Protocol มีหน้าที่เลือกเส้นทางที่ใช้รับส่งข้อมูลผ่านระบบเครือข่าย และตรวจสอบแอดเดรส (Address) ของผู้รับโดยใช้ข้อมูลขนาด 4 ไบต์หรือ 32 บิต เป็นตัวกำหนดแอดเดรส (Address) ของผู้รับเรียกว่าไอพีแอดเดรส (IP Address)

2.1 โครงสร้างของโปรโตคอลทีซีพี/ไอพี (TCP/IP)

โปรโตคอลทีซีพี/ไอพี (TCP/IP) มีกลไกการทำงานเป็นชั้นหรือเลเยอร์ (layer) เรียงต่อกัน โดยในแต่ละเลเยอร์ (layer) จะมีการทำงานเทียบได้กับโอเอสไอโมเดล (OSI model) แต่บางเลเยอร์ (layer) ของ โปรโตคอลทีซีพี/ไอพี (TCP/IP) จะทำงานเทียบกับโอเอสไอ (OSI) หลายเลเยอร์ ปนกัน ซึ่งในแต่ละ เลเยอร์ ของโปรโตคอล TCP/IP จะประกอบด้วย

- โปรเซสเลเยอร์ (Process layer)
- โฮสต์ทูโฮสต์เลเยอร์ (Host-to-Host layer)
- อินเตอร์เน็ตเวิร์กเลเยอร์ (Internetwork layer)
- เน็ตเวิร์กอินเตอร์เฟซเลเยอร์ (Network Interface layer)

โดยเมื่อเทียบกับโอเอสไอโมเดล (OSI model) แล้วจะเป็นดังรูปที่ 2.1 ซึ่งเราจะเห็นว่าบางกลไกของโปรโตคอลทีซีพี/ไอพี (TCP/IP) เทียบได้กับโอเอสไอโมเดล (OSI model) สองชั้นหรือบางกลไกก็จะทำงานคาบเกี่ยวกันระหว่างบางชั้นของโอเอสไอโมเดลตัวอย่างเช่นกลไกการทำงานของโปรโตคอลทีซีพี/ไอพีในส่วนเน็ตเวิร์กอินเตอร์เฟซเลเยอร์ (Network Interface layer) เมื่อเทียบกับ โอเอสไอโมเดล จะเทียบได้กับดาต้าลิงค์เลเยอร์ (Data Link layer)และ ฟิสิคอลเลเยอร์ (Physical layer) 2 ชั้นรวมกัน เป็นต้น ในแต่ละกลไกของโปรโตคอล ทีซีพี/ไอพี จะมีโปรโตคอลอื่น ๆ ในชุดของ ทีซีพี/ไอพี ร่วมทำงานอยู่ด้วย ซึ่งจะกล่าวโดยละเอียดต่อไป



รูปที่ 2.1 ทีซีพี/ไอพี สแตค เปรียบเทียบกับโอเอสไอโมเดล

(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

2.1.1 โปรเซสเลเยอร์ (Process layer)

จากรูปแสดงลำดับชั้นการทำงานของโปรโตคอล ทีซีพี/ไอพี เทียบกับมาตรฐานโอเอสไอโมเดล นั้น ในชั้นบนสุดเรียกว่าโปรเซสเลเยอร์ (Process layer) ทำงาน 2 หน้าที่เทียบได้กับแอปพลิเคชันเลเยอร์ (Application layer) และพรีเซนต์เตชันเลเยอร์ (Presentation layer) ในชั้นนี้จะรองรับการทำงานของแอปพลิเคชันต่าง ๆ ที่ทำงานเป็นโปรเซส อยู่ในเครื่องเซิร์ฟเวอร์ให้บริการและเครื่องที่ขอใช้บริการหรือไคลเอนต์ (client) ซึ่งจะติดต่อกันผ่านโปรโตคอลเฉพาะแอปพลิเคชันอีกทีหนึ่ง ตัวอย่างเช่น เมื่อผู้ใช้งานอินเตอร์เน็ตต้องการถ่ายโอนข้อมูลหรือดาวน์โหลด (download) ข้อมูลจากเครื่องเซิร์ฟเวอร์ที่ให้บริการโดยอาจจะเรียกใช้โปรแกรมเอฟทีพีไคลเอนต์ (ftp client) ทั่วไป เช่น โปรแกรม WS_ftp ติดต่อกับโปรเซสเอฟทีพี (ftp) ที่กำลังให้บริการอยู่ที่เครื่องเซิร์ฟเวอร์ จากนั้นตัวโปรเซส เอฟทีพี (ftp) ก็จะเรียกให้โปรโตคอลเอฟทีพี (FTP :File Transfer Protocol) เพื่อทำการโอนถ่ายไฟล์นี้ หรือถ้าผู้ใช้ต้องการเรียกใช้งานคอมพิวเตอร์ที่เครื่องที่อยู่ห่างไกลออกไปด้วยการใช้โปรแกรม เทลเน็ต (telnet) ที่เครื่องเซิร์ฟเวอร์ให้บริการ ตัวโปรเซสเทลเน็ต (telnet) ที่ทำงานอยู่ก็จะเรียกใช้โปรโตคอลเทลเน็ต(Telnet)เพื่อติดต่อกันหรือในกรณีที่มีการเรียกใช้โปรแกรมเว็บเบราเซอร์ (web browser) เช่น เน็ตเคปเนวิเกเตอร์ (Netcape Navigator) เพื่อเรียกดูเว็บเพจในเว็บไซต์ ซีเอ็นเอ็น (CNN) ที่เครื่องซึ่งให้บริการเว็บของซีเอ็นเอ็น (CNN) ก็จะมีโปรเซสเอชทีทีพี (HTTP: HyperText Transfer Protocol) ทำงานอยู่และจะติดต่อกับผู้ใช้ผ่านโปรโตคอลเอชทีทีพี (HTTP) เป็นต้น

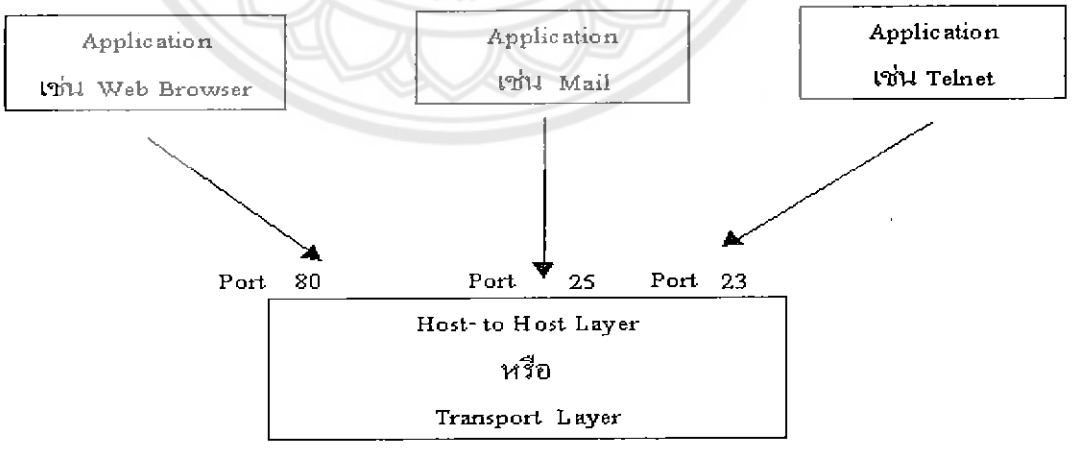
การทำงานของแอปพลิเคชันต่าง ๆ จะอยู่ที่โปรเซสเลเยอร์ (Process layer) นี้ และมีการติดต่อกันตามแต่ละโปรโตคอลเฉพาะ แล้วแต่แอปพลิเคชันที่ใช้งาน จากการที่ โปรเซสเลเยอร์ของ ทีซีพี/ไอพี รองรับให้โปรโตคอลอื่นทำงานได้หลายโปรเซสและหลายโปรโตคอลได้พร้อมกัน นั้นทำให้ผู้ใช้สามารถเปิดโปรแกรมใช้งานได้หลายๆอย่างพร้อมกัน เช่น เปิดโปรแกรมอินเตอร์เน็ตเอ็กซ์พลอเรอร์ (Internet Explorer)เพื่อเรียกดูเว็บเพจพร้อมกับใช้งานโปรแกรมเอาต์ลุค เอ็กซ์เพลส (Outlook Express) เพื่อรับส่งอีเมล์ไปพร้อมกันได้โดยไม่ต้องรอให้ทำงานอย่างหนึ่งอย่างใดเสร็จก่อน หรือในปัจจุบันมีการพัฒนาโปรแกรมเว็บเบราเซอร์ (web browser) ให้สามารถเรียกใช้งานโปรโตคอลอื่น ๆ ได้มากขึ้น ทำให้เราสามารถใช้โปรแกรมเว็บเบราเซอร์ (web browser) โอนถ่ายไฟล์ข้อมูลที่ใช้โปรโตคอลเอฟทีพี (FTP) ได้โดยไม่ต้องไปหาโปรแกรมอื่นมาใช้

โปรโตคอลหลักๆทำงานในโปรเซสเลเยอร์ได้แก่เอฟทีพี(FTP:File TransferProtocol), เอชทีทีพี(HTTP :HyperText Transfer Protocol),เทลเน็ต (Telnet) และเอชเอ็มทีพี (SMTP: Simple Mail Transfer Protocol) นอกจากนี้ยังมีโปรโตคอลอื่นที่อยู่เบื้องหลัง ซึ่งทำงานโดยที่ผู้ใช้ไม่สามารถมองเห็นได้จากโปรแกรมหรือไม่ได้มีการใช้งานโดยตรง เช่น

- โปรโตคอลดีเอ็นเอส (DNS :Domain Name System) ทำหน้าที่แปลงข้อมูลชื่อโดเมนเนม (domain name) หรือชื่อเว็บไซต์ทั้งหลายให้เป็นหมายเลยไอพีแอดเดรส (IP Address)
- โปรโตคอลเอสเอ็นเอ็มพี (SNMP :Simple Network Management Protocol) ใช้ในการควบคุมและตรวจสอบอุปกรณ์ที่อยู่ในเครือข่าย
- โปรโตคอลดีเอชซีพี (DHCP :Dynamic Host Configuration Protocol) ทำหน้าที่แจกจ่ายข้อมูลพารามิเตอร์ของเครือข่ายให้กับเครื่องลูกข่ายที่เชื่อมต่ออยู่

2.1.2 โฮสต์ทูโฮสต์เลเยอร์ (Host-to-Host layer)

การทำงานที่ชั้นของ โฮสต์ทูโฮสต์เลเยอร์ นี้จะมีบทบาทในการจัดการต่อจากโปรเซสเลเยอร์ บางครั้งเรามักเรียกชั้นโฮสต์ทูโฮสต์ (Host-to-Host) ว่าเป็นทรานสปอร์ตเลเยอร์ (Transport layer) ซึ่งไม่ใช่ชั้นของทรานสปอร์ตเลเยอร์ (Transport layer) ในโอเอสไอโมเดลโมเดล การทำงานของ โฮสต์ทูโฮสต์เลเยอร์ นี้ จะมีการสร้างคอนเนคชัน (connection) หรือการเชื่อมต่อกันระหว่างแอปพลิเคชันกับ โฮสต์ทูโฮสต์เลเยอร์ โดยจุดที่เชื่อมเพื่อรับส่งข้อมูลนี้เรียกว่า พอร์ต หรือ ซ็อกเก็ต (socket) (คำว่าพอร์ตในที่นี้ไม่ได้หมายถึง พอร์ต ทางฮาร์ดแวร์)และในแต่ละแอปพลิเคชันก็จะสร้างการเชื่อมต่อผ่าน พอร์ต ได้พร้อมกันหลายแอปพลิเคชัน ซึ่งในการใช้งานพอร์ตของแต่ละแอปพลิเคชันที่อยู่ในชั้นโปรเซสเลเยอร์จะแตกต่างกันตามหมายเลขที่กำหนดไว้และแต่ละโปรโตคอลจะมีการใช้งานพอร์ต หมายเลขต่าง ๆ ไม่ซ้ำกัน ดังรูปที่ 2.2

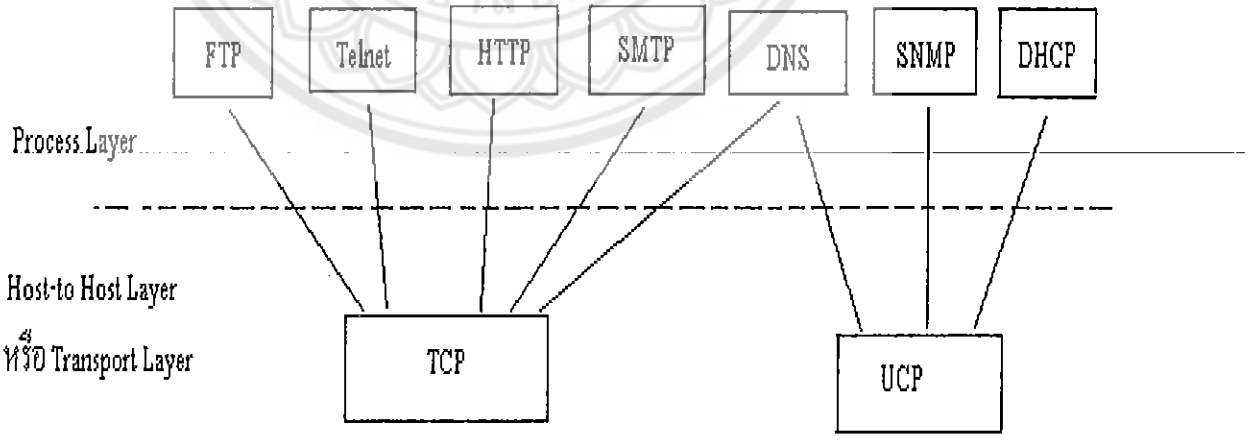


รูปที่ 2.2 แอปพลิเคชันหรือโปรเซสต่าง ๆ สื่อสารกับ โฮสต์ทูโฮสต์เลเยอร์ ผ่านจุดเชื่อมต่อหรือพอร์ต

(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

เมื่อแอปพลิเคชันทำงานผ่านโปรโตคอลในชั้น โปรเซสเลเยอร์ จะมีการส่งผ่าน โฮสต์ทูโฮสต์เลเยอร์ ที่ชั้นนี้จะมีการเชื่อมต่อผ่าน พอร์ต ที่กำหนดทำให้การรับส่งข้อมูลในแต่ละโปรโตคอลทำได้ถูกต้อง ถึงแม้ว่าในเครื่องเซิร์ฟเวอร์ที่ให้บริการจะมีการทำงานอยู่หลายโปรเซสที่แตกต่างกันก็ตามหรือมีผู้ใช้บริการเข้ามาใช้งานพร้อมกันจำนวนมากและหลายแอปพลิเคชันในเวลาเดียวกัน ในชั้น โฮสต์ทูโฮสต์ (Host-to-Host) หรือทรานสปอร์ตเลเยอร์ (Transport layer) ของ ทีซีพี/ไอพี นี้จะมีโปรโตคอลทำงานอยู่ 2 โปรโตคอลที่แตกต่างกัน คือ โปรโตคอลทีซีพี (TCP) และโปรโตคอลยูดีพี (UDP :User Datagram Protocol) ในการส่งผ่านข้อมูลลงไปที่ชั้นถัด ๆ ไป เราจะเห็นว่าโปรโตคอลทีซีพี (TCP) และ ยูดีพี (UDP) จะถูกผลึกเข้าไปในโปรโตคอลไอพี (IP) อีกทีหนึ่ง และส่งต่อไปยังเครือข่ายอินเตอร์เน็ตต่อไป

ตัวโปรโตคอลทีซีพี(TCP) และโปรโตคอลยูดีพี (UDP) จะมีแอปพลิเคชันเฉพาะเพื่อใช้งานแยกกันคือ แอปพลิเคชันที่ใช้โปรโตคอลเอฟทีพี(FTP),เทลเน็ต (Telnet), เอชทีทีพี (HTTP) และเอชเอ็มทีพี (SMTP) จะมีการส่งผ่านข้อมูลโดยเรียกใช้โปรโตคอลทีซีพี (TCP) ส่วนแอปพลิเคชันที่ใช้โปรโตคอล เอสเอ็นเอ็มพี (SNMP) และดีเอชซีพี (DHCP) จะส่งผ่านข้อมูลโดยเรียกใช้โปรโตคอลยูดีพี (UDP) และสำหรับโปรโตคอลดีเอ็นเอส (DNS) นั้น จะสามารถเรียกใช้งานทั้งทีซีพี และยูดีพี ดังรูป ซึ่งมีเหตุผลที่มีการเรียกใช้โปรโตคอลทีซีพี และ ยูดีพี แตกต่างกัน ก็เนื่องจากวิธีการทำงานของทั้งสองโปรโตคอลต่างกันนั่นเอง



รูปที่ 2.3 โปรเซสต่าง ๆ ที่เรียกใช้ทรานสปอร์ตเลเยอร์ (Transport layer)
เพื่อส่งผ่านข้อมูลโดยอาศัย พอร์ต
(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

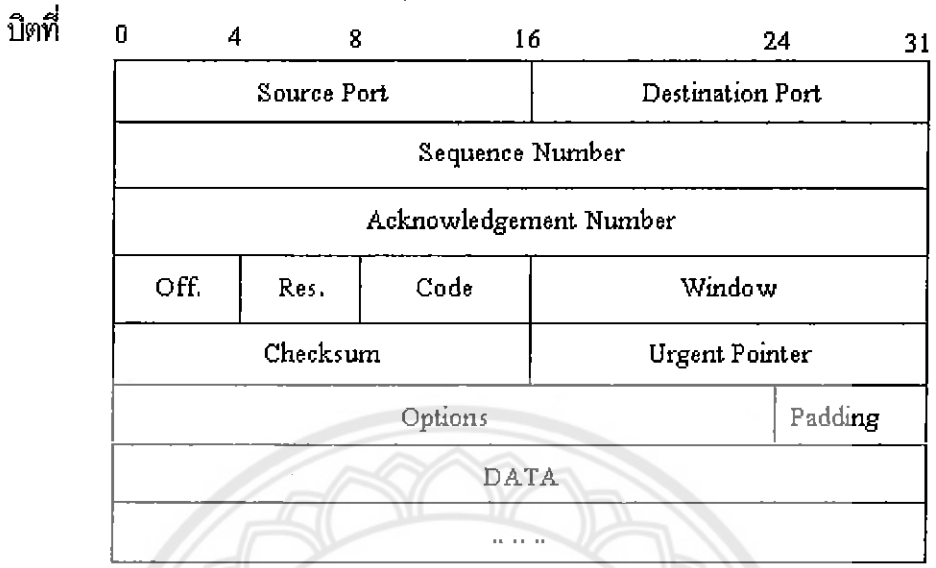
2.1.2.1 โปรโตคอลทีซีพี (TCP:Transmission Control Protocol)

เป็นโปรโตคอลที่มีการรับส่งข้อมูลแบบ stream oriented protocol หมายความว่า การรับส่งข้อมูลจะไม่คำนึงถึงปริมาณข้อมูลที่จะส่งไป แต่ละแบ่งข้อมูลเป็นส่วนย่อย ๆ ก่อนแล้วจึงจะส่งไปยังปลายทางอย่างต่อเนื่องเป็นลำดับข้อมูล ในกรณีที่ข้อมูลส่วนใดส่วนหนึ่งสูญหายไปก็จะส่งข้อมูลส่วนนั้นใหม่อีกครั้งสำหรับปลายทางก็จะทำหน้าที่จัดเรียงส่วนของข้อมูลดาต้าแกรม (datagram) ใหม่ให้ต่อเนื่องกัน และประกอบกลับเป็นข้อมูลทั้งหมดได้ ซึ่งจะแยกข้อมูลที่ไม่ถูกต้องออก ดังนั้นแอปพลิเคชันหรือโปรเซสใดที่อาศัยการส่งผ่านข้อมูลด้วยโปรโตคอลทีซีพี จะต้องใช้หน่วยความจำหรือขนาดของช่องสัญญาณ (bandwidth) มากกว่ายูดีพี

การติดต่อระหว่างกันจะต้องเป็นแบบคอนเนคชันออเรียนเตต (connection-oriented) คือ ต้องมีการสร้างการติดต่อกันเป็นเซสชัน (session) ทั้ง 2 ด้านเสียก่อน แล้วจึงจะรับส่งข้อมูลไปได้พร้อมกัน (full duplex) เหมือนกับการใช้โทรศัพท์ติดต่อกัน เมื่อผู้ติดต่อต้นทางเรียกให้ฝ่ายตรงข้ามรับสายแล้ว จึงเริ่มการสนทนา เช่น พูดคำว่า "สวัสดี" หรือ "ฮัลโหล" กันก่อนเพื่อให้แน่ใจว่าฝ่ายตรงข้ามพร้อมจะติดต่อด้วย จากนั้นจึงเริ่มต้นติดต่อกัน และเมื่อต้องการจะเลิกการติดต่อก็จะมีการพูดคำว่า "สวัสดี" ให้ฝ่ายตรงข้ามทราบว่าจะเลิกการติดต่อและวางสายไป ซึ่งในระหว่างการติดต่อกันนั้น แม้ว่าฝ่ายใดฝ่ายหนึ่งหรือทั้งสองฝ่ายจะเงียบไป คือ ไม่พูดอะไรเป็นเวลานาน ๆ แต่การเชื่อมโยงระหว่างทั้งสองด้านยังคงมีอยู่ไม่ขาดไปจนกว่าฝ่ายใดฝ่ายหนึ่งจะวางสาย เช่นเดียวกับการติดต่อกันด้วยกลไกโปรโตคอลทีซีพี เมื่อแอปพลิเคชันต้องการส่งผ่านข้อมูลจะใช้โปรโตคอลที่เหมาะสมในชั้น โปรเซสเลเยอร์ติดต่อไป และมีการสร้างช่องข้อมูลผ่าน พอร์ต ที่กำหนดเพื่อส่งผ่านข้อมูลไปยังโปรโตคอล ทีซีพี

ในระหว่างการรับส่งข้อมูลนี้ โปรโตคอลทีซีพี จะเพิ่มขบวนการสอบทานข้อมูล เพื่อให้ข้อมูลมีความถูกต้องไม่ผิดพลาดไปจากเดิมโดยการส่งสัญญาณสอบทานข้อมูล (acknowledgement) และส่งข้อมูลให้ใหม่อีกครั้ง ถ้าปลายทางไม่ได้รับหรือเกิดความผิดพลาดขึ้น

ความน่าเชื่อถือของการส่งผ่านข้อมูลโดยโปรโตคอลทีซีพีจะมีมากกว่าแต่ก็ต้องอาศัยทรัพยากรของระบบมากกว่าในการทำงานเช่นกัน



รูปที่ 2.4 รูปแบบของทีซีพีแพคเก็ต (TCP packet) จะเห็นว่ามีฟิลด์ Acknowledgement Number และข้อมูลเช็คซัม (Checksum)
(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

2.1.2.2 โปรโตคอลยูดีพี (UDP Protocol)

ในโฮสต์ทูโฮสต์เลเยอร์นอกจากจะมีโปรโตคอลทีซีพีทำงานแล้วก็ยังมีโปรโตคอลยูดีพี (User Datagram Protocol) ที่มีคุณสมบัติแตกต่างกันอยู่ด้วยในการรับส่งข้อมูลผ่านโปรโตคอลยูดีพีจะเป็นแบบที่ทั้งสองด้านไม่จำเป็นต้องอาศัยการสร้างช่องทางเชื่อมต่อกัน (connectionless) ระหว่างเครื่องเซิร์ฟเวอร์ให้บริการกับเครื่องที่ขอใช้บริการ โดยไม่ต้องแจ้งให้ฝ่ายรับข้อมูลเตรียมรับข้อมูลเหมือนโปรโตคอลทีซีพี และไม่มีการตรวจสอบความถูกต้องครบถ้วนในการรับส่งข้อมูลนั้นๆด้วยเนื่องจากโปรโตคอลยูดีพีไม่มีสัญญาณสอบทานข้อมูล (acknowledgement) ในการส่งข้อมูลแต่ละครั้ง และไม่มีการส่งข้อมูลใหม่อีกในกรณีที่เกิดความผิดพลาดของการส่งข้อมูล เมื่อเป็นเช่นนี้แอปพลิเคชันหรือโปรเซสใดที่ต้องอาศัยโปรโตคอล ยูดีพี ในการส่งผ่านข้อมูลก็อาจจะต้องสร้างขบวนการตรวจสอบข้อมูลขึ้นมาเอง

ตามรูป 2.3 จะเห็นว่าโปรโตคอลชั้นบนขึ้นไป ที่ใช้การส่งผ่านข้อมูลโดยโปรโตคอลยูดีพี เช่น โปรโตคอลเอสเอ็นเอ็มพี (SNMP:ใช้ควบคุมและจัดการอุปกรณ์ในเครือข่าย), หรือโปรโตคอลดีเอชซีพี (DHCP:ใช้ส่งข้อมูลพารามิเตอร์ของเครือข่ายให้กับเครื่องลูกข่ายได้ใช้งาน) การส่งข้อมูล

เหล่านั้นไม่ต้องรับทราบหรือตรวจสอบว่าข้อมูลไปถึงปลายทางถูกต้องหรือไม่ แต่กลไกการตรวจสอบข้อมูลที่มีการรับส่งจะไปทำในขั้นตอนของโปรโตคอลชั้นที่สูงกว่าแทน

ตัวอย่างขั้นตอนกลไกการทำงานโดยใช้โปรโตคอลยูดีพี มีดังต่อไปนี้

1. ในชั้นของโปรเซสเลเยอร์ เมื่อโปรแกรมควบคุมอุปกรณ์เครือข่าย เช่น โปรแกรมเน็ตเวิร์คเมเนเจอร์ (Network management) ต้องการส่งข้อมูลไปยังอุปกรณ์ที่ต้องการแอปพลิเคชันนั้น จะติดต่อผ่านโปรโตคอลเอสเอ็นเอ็มพี (SNMP) ในชั้นโปรเซสเลเยอร์

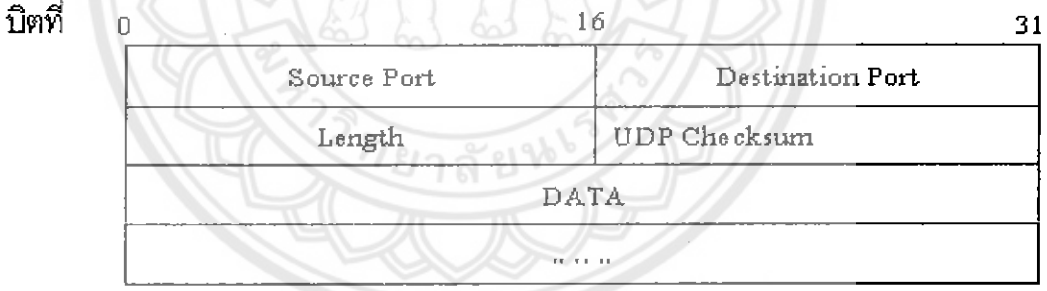
2. โปรโตคอลเอสเอ็นเอ็มพี จะติดต่อกับโปรโตคอล ยูดีพี ในชั้นถัดไป เพื่อขอติดต่อผ่านพอร์ต ที่กำหนด

3. โปรโตคอลเอสเอ็นเอ็มพี เตรียมข้อมูลที่จะส่ง รวมทั้งที่อยู่ปลายทาง

4. โปรโตคอลเอสเอ็นเอ็มพี ส่งผ่านข้อมูลให้โปรโตคอลยูดีพีที่อยู่ในโฮสต์ทูโฮสต์เลเยอร์

5. โปรโตคอลยูดีพี ทำหน้าที่ผนึกข้อมูลหรือดาต้าแกรม (datagram) นั้น ไปกับ โปรโตคอลไอพี (IP) ในชั้นถัดลงไป เพื่อส่งข้อมูลออกจากเครื่อง

ซึ่งจะเห็นว่ามีกลไกที่ต่างจากการส่งข้อมูลด้วยโปรโตคอลทีซีพี ซึ่งจะต้องมีการติดต่อกันก่อน และทั้งสองฝ่ายรับทราบการรับส่งข้อมูลของช่องการส่งข้อมูลนั้น



รูปที่ 2.5 รูปแบบของยูดีพี แพคเก็ต จะมีฟิลด์ข้อมูลส่วนเฮดเดอร์ น้อยมากและไม่มีส่วนการตรวจ สอบข้อมูล

(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

2.1.3 อินเตอร์เน็ตเวิร์กเลเยอร์ (Internetwork Layer)

ในระดับล่างต่อมาในชั้นอินเตอร์เน็ตเวิร์กเลเยอร์ (Internetwork Layer) มีหน้าที่ส่งผ่านข้อมูลในระหว่างเครือข่าย โดยมีโปรโตคอลที่ทำงานเป็นกลไกสำคัญในการส่งผ่านข้อมูลไปยังเครือข่ายใด ๆ บนอินเตอร์เน็ต คือ โปรโตคอลไอพี (IP :Internet Protocol) นอกจากนี้ในชั้นอินเตอร์เน็ตเวิร์กเลเยอร์ (Internetwork Layer) ยังมีโปรโตคอลทำงานอยู่ด้วยกัน2ชนิด คือ

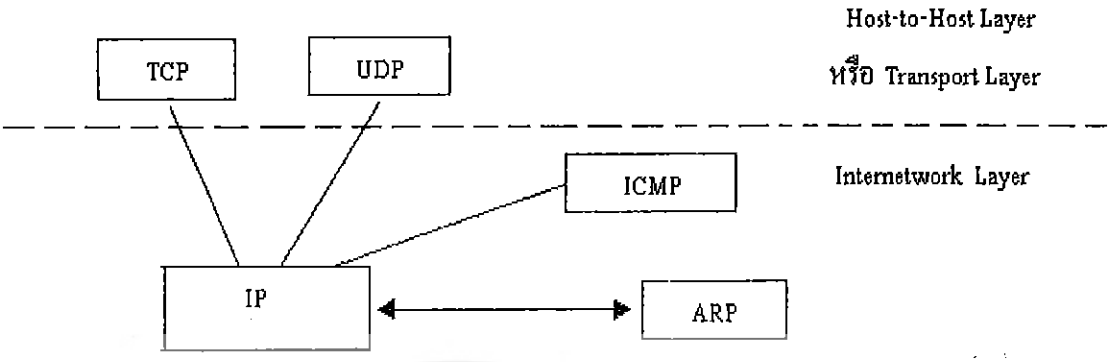
โปรโตคอลไอซีเอ็มพี (Internet Control Message Protocol : ICMP) และโปรโตคอลเออาร์พี (Address Resolution Protocol :ARP)

2.1.3.1 โปรโตคอลไอพี (IP)

โปรโตคอลไอพี(IP)ทำหน้าที่ให้บริการส่งผ่านข้อมูลที่มาจากโฮสต์ทูโฮสต์เลเยอร์ เพื่อส่งข้ามไปยังเครือข่ายใด ๆ ได้อย่างถูกต้อง แม้ว่าจะมีเครือข่ายเชื่อมต่อกันอยู่ในอินเตอร์เน็ตเป็นล้าน ๆ เครือข่ายก็ตาม เนื่องจากโปรโตคอลไอพี มีข้อมูลตำแหน่ง ไอพี ปลายทางที่จะส่งไปให้ โดยทำงานร่วมกับอุปกรณ์เราเตอร์ (Router) เพื่อส่งข้อมูลข้ามเครือข่ายออกไปได้ ตัวโปรโตคอลไอพี จะทำงานแบบแพคเก็ตสวิทชิ่ง (packet switching) คือมีการส่งข้อมูลผ่านสวิตช์(switch) ไปยังปลายทาง โดยข้อมูลจะเดินทางไปยังเครือข่ายต่าง ๆ ผ่านสวิตช์นี้ไปเรื่อย ๆ จนกว่าจะถึงปลายทาง ตัววงจรผ่านหรือสวิตช์ (switch) นี้อาจเป็นเกทเวย์ (Gateway) หรือเราเตอร์ (Router) ในระบบเครือข่ายก็ได้ ซึ่งในข้อมูลของโปรโตคอลไอพี จะมีข้อมูลหมายเลขไอพี ปลายทางที่จะส่งข้อมูลไป และเมื่อถึงเครือข่ายปลายทางแล้ว จะมีกลไกแปลงหมายเลข ไอพี ให้เป็นหมายเลขฮาร์ดแวร์ประจำเครื่องที่ถูกต้องอีกทีหนึ่งด้วยโปรโตคอลเออาร์พี (ARP) ตามรูปที่ 2.6 ที่จะแสดงการติดต่อกันระหว่างโปรโตคอลในชั้นของโฮสต์ทูโฮสต์เลเยอร์ และ อินเตอร์เน็ตเวิร์กเลเยอร์

2.1.3.2 โปรโตคอลไอซีเอ็มพี (ICMP)

หน้าที่หลักของโปรโตคอลไอซีเอ็มพี (ICMP :Internet Control Message Protocol) คือ การแจ้งหรือแสดงข้อความจากระบบ เพื่อบอกให้ผู้ใช้ทราบว่าเกิดอะไรขึ้นในการส่งผ่านข้อมูลนั้น ซึ่งปัญหาส่วนมากที่พบคือส่งไปไม่ได้ หรือปลายทางรับข้อมูลไม่ได้ เป็นต้น นอกจากนี้โปรโตคอลไอซีเอ็มพี (ICMP) ยังถูกเรียกใช้งานจากเครื่องเซิร์ฟเวอร์และเราเตอร์ (Router) อีกด้วย เพื่อแลกเปลี่ยนข้อมูลที่ใช้ควบคุม ส่วนรูปแบบการทำงานของโปรโตคอลไอซีเอ็มพี (ICMP) นั้นจะทำควบคู่กับโปรโตคอล ไอพี ในระดับเดียวกัน และข้อความต่าง ๆ ที่แจ้งให้ทราบจะถูกผนึกอยู่ภายในข้อมูลของไอพี (IP datagram) อีกทีหนึ่ง



รูปที่ 2.6 โปรโตคอลทีซีพีและยูดีพี อาศัยโปรโตคอลไอพี
ที่อยู่ชั้นล่างเพื่อส่งผ่านข้อมูลระหว่างเครือข่าย
(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

ข้อความที่โปรโตคอลไอซีเอ็มพี (ICMP) ส่งนั้นแบ่งได้ 2 แบบ คือไอซีเอ็มพีเมซเซส (ICMP error message) หรือข้อความแจ้งข้อผิดพลาด และไอซีเอ็มพีคิวรี (ICMP query) หรือข้อความเรียกขอข้อมูลเพิ่ม ตัวอย่างกลไกการทำงานของโปรโตคอลไอซีเอ็มพี(ICMP)ที่ชื่อเดสทิเนชันอันรีชเอเบิล (destination unreachable) ให้กับผู้ส่งข้อมูล นอกจากนี้ตัวข้อมูลที่แจ้งข้อความก็จะมีส่วนของข้อมูล ไอพี ดาต้าแกรม (IP datagram) ที่เกิดปัญหาด้วย ดังนั้นเมื่อผู้ส่งข้อมูลได้รับข้อความแจ้งก็จะทราบได้ว่าจุดที่เกิดปัญหานั้นอยู่ที่ใด

ดังนั้นโปรโตคอลไอซีเอ็มพี (ICMP) จึงกลายมาเป็นเครื่องมืออย่างหนึ่งในการช่วยทดสอบเครือข่าย เช่น คำสั่ง ping ที่เราใช้ทดสอบว่าเครื่องเซิร์ฟเวอร์ที่ให้บริการหรืออุปกรณ์ที่อยู่ในระบบเครือข่ายอินเตอร์เน็ตนั้นยังทำงานเป็นปกติหรือไม่

2.1.3.3 โปรโตคอลเออาร์พี (ARP)

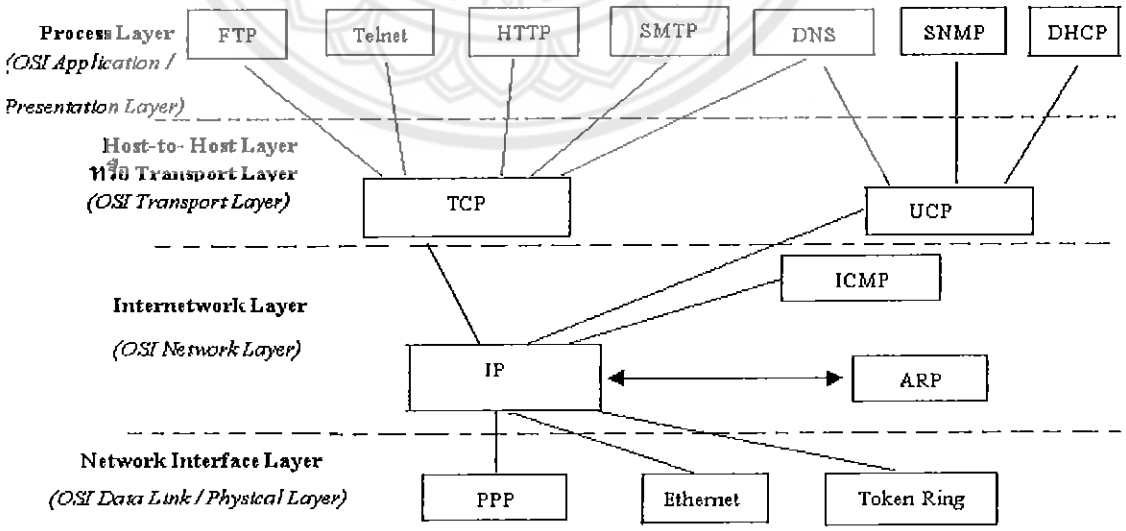
โปรโตคอลเออาร์พี (ARP :Address Resolution Protocol) ถูกเรียกใช้งานโดยโปรโตคอลไอพี เพื่อช่วยแปลงหมายเลขไอพี ไปเป็นหมายเลขฮาร์ดแวร์ปลายทาง ตัวอย่างเช่น เว็บเซิร์ฟเวอร์เครื่องหนึ่งเชื่อมต่ออยู่ในเครือข่ายอินเตอร์เน็ต และในการเชื่อมต่อนี้ต้องอาศัยเน็ตเวิร์กอินเตอร์เฟซการ์ด (Network Interface Card :NIC) หรือแลนการ์ด (LAN card) ติดตั้งอยู่ที่แลนการ์ด (LAN card) นี้เองจะมีหมายเลขเฉพาะประจำฮาร์ดแวร์ที่ไม่ซ้ำกับใคร เพื่อใช้อ้างอิงการส่งข้อมูลในเครือข่าย แต่เมื่อมาใช้งานโปรโตคอลทีซีพี/ไอพี ก็จะต้องมีการกำหนดหมายเลข

ไอพีแอดเดรส ประจำตัวเพื่อใช้อ้างอิงกัน และโปรโตคอลเออาร์พี (ARP) จะทำหน้าที่แปลงค่าหมายเลข ไอพี ให้เป็นหมายเลขฮาร์ดแวร์ให้ในระดับการทำงานที่ อินเตอร์เน็ตเวิร์กเลเยอร์ (Internetwork layer) นี้ ซึ่งกลไกการแปลงนี้เรียกว่า แอดเดรส รีโซลูชัน (address resolution)

2.1.4 เน็ตเวิร์กอินเตอร์เฟซเลเยอร์ (Network Interface Layer)

เนื่องจากในด้านกายภาพของเครือข่ายนั้น มีหลายวิธีการและหลายรูปแบบในการเชื่อมต่อระบบให้เป็นเครือข่าย แต่อย่างไรก็ตามในเครือข่ายอินเตอร์เน็ตนี้ ข้อมูลหรือไอพี ดาต้าแกรม (IP datagram) จะถูกถ่ายทอดและส่งผ่านไปยังปลายทางโดยไม่คำนึงถึงรูปแบบการเชื่อมต่อทางกายภาพ ไม่ว่าจะเป็นการใช้เครือข่ายใยแก้วนำแสงหรือเครือข่ายสาย Unshielded Twist Pair (UTP) เชื่อมต่อเป็นแบบเครือข่ายอีเธอร์เน็ต (Ethernet) ธรรมดาหรือเครือข่ายโทคเก้นริง (Token Ring), เอทีเอ็ม (ATM), ไอเอสดีเอ็น (ISDN) ฯลฯ ก็ตาม

การทำงานระดับล่างสุดต่อจากอินเตอร์เน็ตเวิร์กเลเยอร์ (Internetwork layer) จะเป็นการแปลงข้อมูลไอพี ดาต้าแกรม (IP datagram) ให้อยู่ในรูปที่เหมาะสม และแปลงเป็นสัญญาณไฟฟ้าส่งไปยังเครือข่ายต่อไป ซึ่งในชั้นเน็ตเวิร์กอินเตอร์เฟซเลเยอร์ (Network Interface layer) นี้ เมื่อเทียบกับโอเอสไอโมเดลโมเดล แล้วจะเป็นการรวม 2 layer เข้าด้วยกันคือดาต้าลิงค์เลเยอร์ (Data Link layer) และฟิสิคอลเลเยอร์ (Physical layer) กล่าวโดยสรุปคือ การทำงานในชั้นต่าง ๆ ตามโครงสร้างของโปรโตคอลทีซีพี/ไอพี จะมีลักษณะดังรูปที่ 2.7



รูปที่ 2.7 โครงสร้างของโปรโตคอล ทีซีพี/ไอพี ในแต่ละชั้นหรือเลเยอร์ (layer)
(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

ตารางที่ 2.1 สรุปหมายเลขบางส่วนของพอร์ตที่ใช้งานโดยทีซีพีและยูดีพี
(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

โปรโตคอลที่ใช้งาน	พอร์ตหรือซ็อกเก็ตเชื่อมต่อ (เลขฐาน 10)	โปรโตคอลในระดับโฮสต์ทูโฮสต์ (Host-to-Host)	รายละเอียด
BootP	67	UDP	BOOTstrap Protocol ด้านเซิร์ฟเวอร์
BootP	68	UDP	BOOTstrap Protocol ด้านไคลเอนต์
DHCP	67	UDP	Dynamic Host Configuration Protocol ด้านเซิร์ฟเวอร์
DHCP	68	UDP	Dynamic Host Configuration Protocol ด้านไคลเอนต์
DNS	53	UDP/TCP	Domain Name System
FTP	21	TCP	File Transfer Protocol ด้านเซิร์ฟเวอร์ที่ควบคุม
FTP	20	TCP	File Transfer Protocol ด้านเซิร์ฟเวอร์ที่ส่งข้อมูล
HTTP	80	TCP/UDP	Hyper Text Transfer Protocol ด้านเซิร์ฟเวอร์
SMTP	25	TCP	Simple Mail Transfer Protocol ด้านเซิร์ฟเวอร์
SNMP	161	UDP	Simple Network Management Protocol ด้าน agent
SNMP	162	UDP	SNMP trap manager
Telnet	23	TCP	Teletype Network Protocol

กล่าวโดยสรุปก็คือ โปรโตคอล TPC/IP ทำงานโดยแบ่งเป็นชั้นเทียบกับ โอเอสไอโมเดล ได้ กลไกในการทำงานของโปรโตคอล ทีซีพี/ไอพี มี 4 ชั้น ซึ่งในขั้นแรก คือ โปรเซสเลเยอร์ ทำหน้าที่ติดต่อกับแอปพลิเคชันและโปรโตคอลที่แอปพลิเคชันนั้นๆใช้งาน และส่งต่อมาให้ชั้น โฮสต์ทูโฮสต์เลเยอร์ เพื่อติดต่อกันระหว่างเครื่องเซิร์ฟเวอร์ให้บริการกับเครื่องผู้ขอใช้บริการ ในชั้นนี้จะมีการสร้างเซสชัน (session) หรือการเชื่อมต่อระหว่างระบบขึ้นตามแต่ละโปรโตคอลต้องการ ต่อมาเป็นการผนึกข้อมูลไปเป็นไอพี ดาต้าแกรม (IP datagram) ที่ชั้นอินเตอร์เน็ตเวิร์กเลเยอร์ (Internetwork layer) โดยอาศัยโปรโตคอล ไอพี เพื่อให้สามารถติดต่อส่งข้อมูลข้ามเครือข่ายไปยังเครือข่ายและเครื่องที่ถูกต้องได้ และสุดท้ายการส่งข้อมูลออกสู่โลกภายนอกต้องอาศัยกลไกชั้นเน็ตเวิร์กอินเตอร์เฟซเลเยอร์ (Network Interface layer) เพื่อแปลงข้อมูลใหม่ เพิ่มข้อมูลที่จำเป็นในการอ้างอิงตำแหน่งและแปลงข้อมูลเป็นสัญญาณไฟฟ้าส่งออกไปยังเครือข่าย และอาจจะออกไปยังเกทเวย์ (Gateway) หรือเราเตอร์ (Router) เพื่อข้ามเครือข่ายออกไปยังเส้นทางที่กำหนดไว้ในอินเตอร์เน็ตต่อไป

เราจะเห็นว่าในแต่ละชั้นของโครงสร้าง ทีซีพี/ไอพี สแตค มีการใช้งานโปรโตคอลต่าง ๆ อยู่หนึ่งโปรโตคอลหรือมากกว่า ในแต่ละโปรโตคอลเหล่านี้ก็จะรับผิดชอบทำหน้าที่ของตน เพื่อส่งผ่านข้อมูลลงไปยังระดับล่าง และออกสู่เครือข่ายอินเตอร์เน็ตในที่สุด

2.1.4.1 กลไกของโปรโตคอลไอพี

ในการส่งผ่านข้อมูลหรือไอพี ดาต้าแกรม (IP datagram) ไปยังเครือข่ายอินเตอร์เน็ตนั้น โปรโตคอล ไอพี ทำหน้าที่พิจารณาว่าปลายทางในการส่งไอพี ดาต้าแกรม (IP datagram) นั้นจะเป็นภายในเครือข่ายของตนเองหรือจะต้องส่งข้อมูลข้ามเครือข่ายไปอีก โดยการพิจารณานี้ โปรโตคอลไอพี จะตรวจสอบจากค่าไอพีแอดเดรส ปลายทางว่าส่วนที่เป็นค่าหมายเลขเครือข่าย (network address) จะเหมือนกับค่าหมายเลขเครือข่ายของ ไอพีแอดเดรส ต้นทางหรือไม่ ถ้าค่าตรงกันแสดงว่าการส่งข้อมูลอยู่ภายในเครือข่ายเดียวกัน แต่ถ้าค่าต่างกัน แสดงว่าต้องส่งข้อมูลไปยังปลายทางที่อยู่คนละเครือข่ายกัน

การส่งข้อมูลภายในเครือข่ายเดียวกัน มีกลไกดังนี้

1. โปรโตคอลไอพี จะเรียกใช้บริการโปรโตคอลเออาร์พี (ARP :Address Resolution Protocol) เพื่อแปลงหมายเลข ไอพี ปลายทางให้เป็นค่าหมายเลขฮาร์ดแวร์ เช่น แมคแอดเดรส (MAC address)

2. เมื่อโปรโตคอลไอพี ได้รับค่าหมายเลขฮาร์ดแวร์แล้ว ก็จะส่งข้อมูลนั้นไปยังฮาร์ดแวร์ที่ระบุไว้

การส่งข้อมูลข้ามเครือข่าย มีกลไกดังนี้

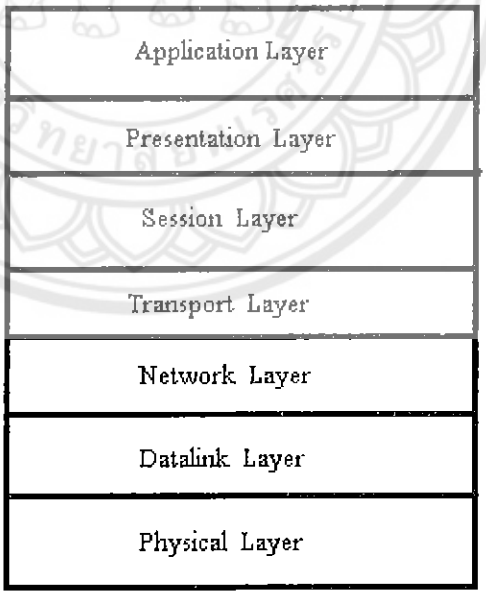
1. โปรโตคอลไอพี ตรวจสอบพบว่าหมายเลขไอพีแอดเดรส ปลายทางอยู่คนละเครือข่ายกัน โดยโปรโตคอลไอพี จะอ่านค่า ไอพีแอดเดรส ของเราเตอร์ (Router) เพื่อเตรียมส่งข้อมูลไปที่ เราเตอร์ (Router) แทน

2. โปรโตคอลไอพีจะเรียกใช้บริการโปรโตคอลเออาร์พี(ARP)เพื่อแปลงค่าไอพีแอดเดรส ของ เราเตอร์ (Router) ให้เป็นค่าหมายเลขฮาร์ดแวร์

3. โปรโตคอลไอพี ส่งข้อมูลไอพี ดาต้าแกรม (IP datagram)ไปยังเราเตอร์ (Router) ที่กำหนดไว้ จากนั้นเราเตอร์ (Router)ส่งข้อมูลข้ามเครือข่ายไปตามขั้นตอนต่อไป

3. โอเอสไอโมเดล (OSI Model) : มาตรฐานอ้างอิงในการสื่อสารข้อมูล

โอเอสไอ (OSI) กำหนดให้การสื่อสารข้อมูลจากระบบคอมพิวเตอร์หนึ่งไปยังอีกระบบหนึ่งแบ่งออกเป็น 7 ขั้นตอนย่อย ๆ ซึ่งคอมพิวเตอร์ทั้งสองระบบจะมีขั้นตอนทั้ง 7 นี้เหมือนทั้งสองฝั่ง เราเรียกการสื่อสารข้อมูลนี้ว่าโอเอสไอเซเว่นเลเยอร์เรฟเฟอร์เรนซ์โมเดล (OSI 7-Layer Reference Model) ดังรูปที่ 2.8

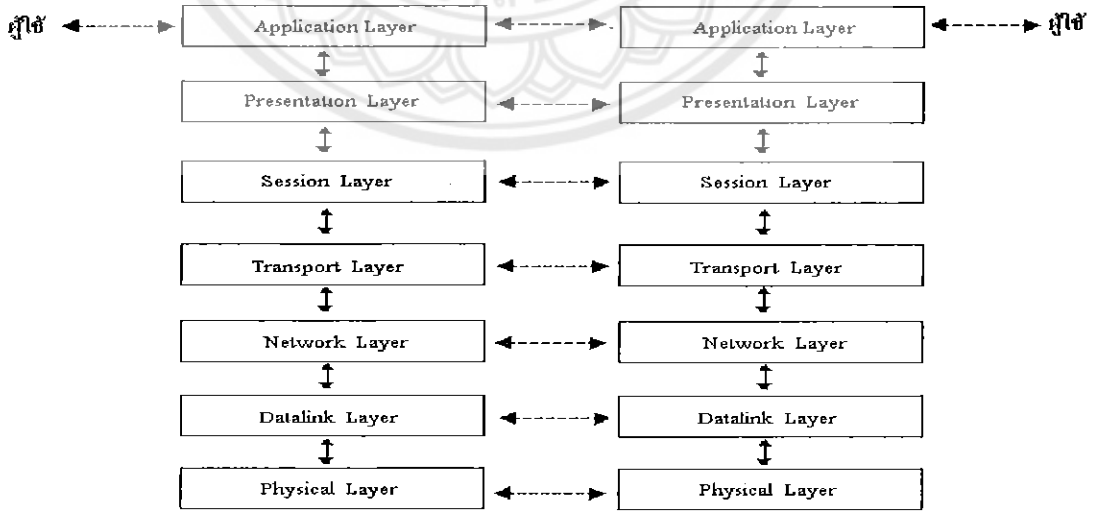


รูปที่ 2.8 โครงสร้างของโอเอสไอโมเดล

(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

โดยโครงสร้างการสื่อสารข้อมูลที่กำหนดขึ้นมีคุณสมบัติดังนี้ คือ แต่ละชั้นของการสื่อสารข้อมูลเราเรียกว่าเลเยอร์ (Layer) หรือ ชั้น ในการสื่อสารข้อมูลจะประกอบด้วยชั้น ๆ ย่อย ๆ 7 ชั้น ในแต่ละชั้นหรือแต่ละเลเยอร์ (Layer) จะเสมือนเชื่อมต่ออยู่กับชั้นที่เทียบเท่ากันของคอมพิวเตอร์อีกด้านหนึ่ง ส่วนการเชื่อมต่อกันจริง ๆ จะมีเพียงชั้นที่ 1 หรือ เลเยอร์ (Layer) 1 ซึ่งเป็นชั้นล่างสุดเท่านั้นที่มีการรับส่งข้อมูลเกิดขึ้นผ่านสายส่งข้อมูลระหว่างคอมพิวเตอร์ทั้งสองระบบ ส่วนชั้นอื่น ๆ จะไม่ได้เชื่อมต่อกันจริง เพียงแต่ทำงานเสมือนกับว่ามีการติดต่อรับส่งข้อมูลกับกลไกในชั้นเดียวกันของคอมพิวเตอร์อีกด้านหนึ่งเท่านั้น

คุณสมบัติข้อที่สองของ OSI 7-Layer Model ก็คือ แต่ละชั้นที่ทำหน้าที่รับส่งข้อมูลจะมีการติดต่อรับส่งข้อมูลกับชั้นที่อยู่ติดกับตัวเองเท่านั้น จะติดต่อรับส่งข้อมูลข้ามกระโดดไปชั้นอื่น ๆ ในคอมพิวเตอร์ของตัวเองไม่ได้ เช่น คอมพิวเตอร์ด้านที่ส่งข้อมูลออกไปให้ผู้รับ ชั้นที่ 7 ซึ่งอยู่บนสุดของด้านส่งข้อมูล จะมีการเชื่อมต่อกับชั้นที่ 6 เท่านั้น ซึ่งชั้นที่ 6 นี้ก็จะมีการเชื่อมต่อรับส่งข้อมูลกับชั้นที่ 7 และชั้นที่ 5 ของคอมพิวเตอร์ด้านส่งข้อมูลเท่านั้น ส่วนชั้นที่ 7 จะกระโดดไปทำการรับส่งข้อมูลกับชั้นที่ 4 หรือ 5 ไม่ได้ การส่งข้อมูลจะไล่ลำดับชั้นลงมาจนถึงชั้นที่ 1 ซึ่งจะเป็นชั้นเดียวที่เชื่อมต่อจริงเข้ากับคอมพิวเตอร์ด้านรับข้อมูลผ่านสายส่งข้อมูล และจะทำการรับข้อมูลจากชั้นที่ 1 ไล่ขึ้นไปจนถึงชั้นที่ 7 ตามลำดับ ลำดับชั้นของการส่งข้อมูลชั้นที่ 7 จะเสมือนเชื่อมต่อเข้ากับลำดับชั้นการรับข้อมูลในชั้นที่ 7 ของคอมพิวเตอร์อีกด้านหนึ่ง ดังแสดงในรูปที่ 2.9

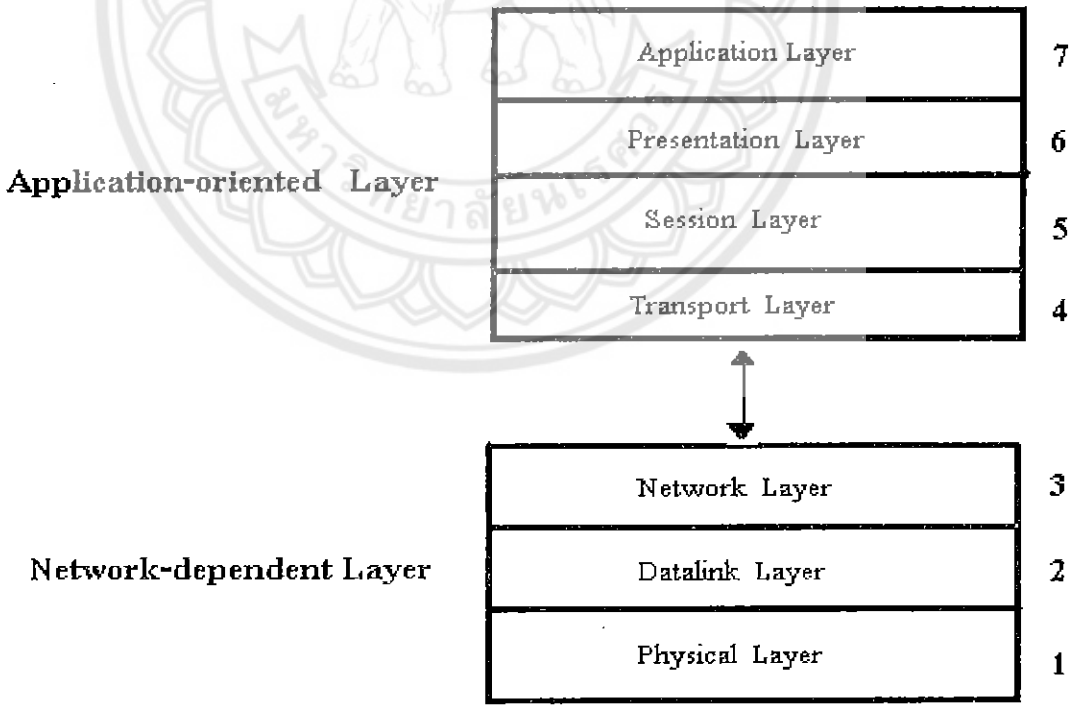


รูปที่ 2.9 การรับส่งข้อมูลของโอเอสไอโมเดล

(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

ผู้ใช้ จะติดต่อรับส่งข้อมูลผ่านทางชั้นที่ 7 ซึ่งอยู่ด้านบนสุดของ โอเอสไอโมเดลเท่านั้น ในทางทฤษฎีแล้วแต่ละชั้นของการรับส่งข้อมูลจะมีฟังก์ชันการทำงานที่แน่นอน และแยกเด็ดขาดออกจากกัน สามารถที่จะนำแต่ละชั้นของแต่ละบริษัทมาเชื่อมต่อกันอย่างไม่มีขีดจำกัด แต่ในทางปฏิบัตินั้นโอเอสไอโมเดล (OSI Model) จะแบ่งออกเป็น 2 กลุ่มใหญ่ ๆ คือ กลุ่มแรกได้แก่ 4 ชั้นด้านบน คือชั้นที่ 7,6,5 และ 4 ทำหน้าที่เชื่อมต่อรับส่งข้อมูลระหว่างผู้ใช้ซอฟต์แวร์โปรแกรมประยุกต์ ให้รับส่งข้อมูลกับฮาร์ดแวร์ที่อยู่ชั้นล่างได้ถูกต้องเรียกว่า แอปพลิเคชันออเรียนเตดเลเยอร์ (Applicaton-oriented layers) ซึ่งจะเกี่ยวข้องกับซอฟต์แวร์เป็นหลัก โดย 4 ชั้นด้านบนนี้มักจะเป็นซอฟแวร์ของบริษัทใดบริษัทหนึ่งรวมอยู่อย่างเบ็ดเสร็จในโปรแกรมเดียว จะแยกออกจากกันเป็นชั้น ๆ เพื่อใช้โปรแกรมของบริษัทอื่นได้ลำบาก หรือในบางกรณีก็ทำไม่ได้เลย

กลุ่มที่สองจะเป็นชั้นล่าง ได้แก่ชั้นที่ 3,2 และ 1 ทำหน้าที่เกี่ยวกับการรับส่งข้อมูลผ่านสายส่ง และควบคุมการรับส่งข้อมูล ตรวจสอบข้อผิดพลาด รวมทั้งเลือกเส้นทางที่ใช้ในการรับส่งข้อมูล ซึ่งจะเกี่ยวข้องกับฮาร์ดแวร์เป็นหลักเรียกว่าเน็ตเวิร์กดีเพนเดนท์เลเยอร์(Network-dependent layers) ดังแสดงในรูปที่ 2.10



รูปที่ 2.10 การแบ่งกลุ่มของโอเอสไอโมเดล (OSI 7 Model)
(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

ซึ่งในส่วนของ 3 ชั้นล่างสุด หรือชั้นที่ 3,2 และ 1 นั้น เนื่องจากเกี่ยวข้องกับฮาร์ดแวร์และโปรแกรมควบคุมฮาร์ดแวร์เป็นหลักทำให้สามารถแยกแต่ละชั้นออกจากกันได้ง่ายและใช้ผลิตภัณฑ์ของต่างบริษัทกันในแต่ละชั้นได้อย่างไม่มีปัญหา

โอเอสไอโมเดล (OSI Model) ที่แบ่งการรับส่งข้อมูลระหว่างคอมพิวเตอร์สองระบบออกเป็น 7 ชั้นนั้น แต่ละชั้นมีชื่อเรียกและหน้าที่การทำงานดังนี้ คือ

3.1 ชั้นที่ 7 แอปพลิเคชันเลเยอร์ (Application Layer)

เป็นชั้นที่อยู่บนสุดของขบวนการรับส่งข้อมูล ทำหน้าที่เชื่อมต่อผู้ใช้เข้ากับระบบคอมพิวเตอร์ โดยรับคำสั่งต่าง ๆ จากผู้ใช้ให้ระบบคอมพิวเตอร์แปลความหมาย และทำงานตามคำสั่งที่ได้รับในระดับโปรแกรมประยุกต์ เช่น แปลความหมายของการกดปุ่มบนเมาส์ให้เป็นคำสั่งในการก๊อปปี้ไฟล์ หรือดึงข้อมูลมาแสดงผลบนจอภาพ เป็นต้น ซึ่งการแปลคำสั่งจากผู้ใช้ส่งให้กับคอมพิวเตอร์รับไปทำงานนี้ จะต้องแปลออกมาถูกต้องตามกฎ (syntax) ที่ใช้ในระบบปฏิบัติการของคอมพิวเตอร์นั้น ๆ ตัวอย่างเช่น ถ้ามีการก๊อปปี้เกินจำนวนที่ระบบปฏิบัติการใช้อยู่ และชื่อไฟล์ต้องประกอบด้วยตัวอักษรตามที่กำหนด ไม่มีตัวอักษรต้องห้ามมาตั้งเป็นชื่อไฟล์ เป็นต้น สิ่งต่าง ๆ เหล่านี้จะเกิดขึ้นในชั้นที่ 7 ของการสื่อสารข้อมูล รวมทั้งฟังก์ชันในการเชื่อมต่อรับส่งข้อมูลระหว่างชั้นที่ 7 กับชั้นที่ 6 ด้วย

3.2 ชั้นที่ 6 พรีเซนเตชันเลเยอร์ (Presentation Layer)

เป็นชั้นที่ทำหน้าที่ตกลงกับคอมพิวเตอร์อีกด้านหนึ่งว่า การรับส่งข้อมูลในระดับโปรแกรมประยุกต์จะมีขั้นตอนและข้อบังคับอย่างไร ข้อมูลที่ทำการรับส่งกันในชั้นที่ 6 นี้ จะอยู่ในรูปแบบของข้อมูลชั้นสูง ซึ่งอยู่ในรูปแบบของคำสั่งที่มีกฎ (syntax) บังคับอย่างแน่นอน เช่น ในการก๊อปปี้ไฟล์ก็จะมีขั้นตอนย่อยประกอบกัน คือสร้างไฟล์ที่กำหนดขึ้นมาเสียก่อน จากนั้นจึงเปิดไฟล์แล้วทำการรับข้อมูลจากปลายทางมาเก็บลงในไฟล์ที่สร้างขึ้นใหม่นี้ โดยเนื้อหาของข้อมูลที่ทำการรับส่งระหว่างกัน ก็คือคำสั่งของขั้นตอนย่อย ๆ ข้างต้นนั่นเอง คำสั่งเหล่านี้จะต้องหมายถึงว่าจะให้ทำอะไรบ้างและถูกต้องตามกฎด้วย นอกจากนี้ในชั้นที่ 6 ยังทำหน้าที่แปลความหมายของคำสั่งที่ได้รับจากชั้นที่ 7 ให้เป็นคำสั่งระดับปฏิบัติการส่งให้ชั้นที่ 5 ต่อไปอีกด้วย

3.3 ชั้นที่ 5 เซสชันเลเยอร์ (Session Layer)

ทำหน้าที่ควบคุม "จังหวะ"ในการรับส่งข้อมูลของคอมพิวเตอร์ทั้งสองด้านที่รับส่งแลกเปลี่ยนข้อมูลกันให้มีความสอดคล้องกัน (Synchronization) และกำหนดวิธีที่ใช้รับส่งข้อมูล เช่น อาจจะเป็นลักษณะสลับกันส่ง (Half-Duplex) หรือรับส่งข้อมูลพร้อมกันทั้งสองด้าน (Full-Duplex) ซึ่งในชั้นที่ 5 นี้จะเป็นชั้นที่ใช้ควบคุมการรับส่งข้อมูลในลักษณะดังกล่าว ข้อมูลที่รับส่งกันในชั้นที่ 5 นี้ จะอยู่ในรูปของ ไดอะล็อก (dialog) หรือประโยคของข้อมูลที่สนทนาโต้ตอบระหว่างกันด้านรับและด้านที่ส่งข้อมูล ไม่ได้มองเป็นคำสั่งอย่างในชั้นที่ 6 เช่น เมื่อผู้รับได้รับข้อมูลส่วนแรกจากผู้ส่ง ก็จะโต้ตอบกลับไปให้ผู้ส่งรู้ว่าได้รับข้อมูลส่วนแรกเรียบร้อยแล้ว และพร้อมที่จะรับข้อมูลส่วนที่สองต่อไป คล้ายกับการสนทนาโต้ตอบกันระหว่างผู้รับกับผู้ส่งนั่นเอง

3.4 ชั้นที่ 4 ทรานสปอร์ตเลเยอร์ (Transport Layer)

ทำหน้าที่เชื่อมต่อการรับส่งข้อมูลระดับสูงของชั้นที่ 5 (ซึ่งมองข้อมูลอยู่ในรูปที่เรียกว่าไดอะล็อก (dialog) หรือประโยคของข้อมูลที่โต้ตอบกัน) มาเป็นข้อมูลที่รับส่งในระดับฮาร์ดแวร์ เช่น แปลงค่าหรือชื่อของคอมพิวเตอร์ในเครือข่ายให้เป็นเน็ตเวิร์กแอดเดรส (network address) พร้อมทั้งเป็นชั้นที่ควบคุมการรับส่งข้อมูลจากปลายด้านส่งถึงปลายด้านรับข้อมูล ให้ข้อมูลมีการไหลลื่นตลอดเส้นทางตามจังหวะที่ควบคุมจากชั้นที่ 5 โดยในชั้นที่ 4 นี้จะเป็นรอยต่อระหว่างการรับส่งข้อมูลของซอฟต์แวร์กับฮาร์ดแวร์ การรับส่งข้อมูลของระดับสูงจะถูกแยกจากฮาร์ดแวร์ที่ใช้รับส่งข้อมูลที่ชั้นที่ 4 นี้ และไม่มีส่วนใดผูกติดกับฮาร์ดแวร์ที่ใช้รับส่งข้อมูลในระดับล่าง ดังนั้นฮาร์ดแวร์และซอฟต์แวร์ที่ใช้ควบคุมการรับส่งข้อมูลในระดับล่างลงไปจากชั้นที่ 4 จึงสามารถสับเปลี่ยนและใช้ข้ามไปมากับซอฟต์แวร์รับส่งข้อมูลในระดับสูงที่อยู่ข้างบน (ตั้งแต่ชั้นที่ 4 ไปจนถึงชั้นที่ 7) ได้ง่าย หน้าที่อีกประการหนึ่งของชั้นที่ 4 คือ การควบคุมคุณภาพของการรับส่งข้อมูลให้มีมาตรฐานในระดับที่ตกลงกันของทั้งสองฝ่าย และการตัดข้อมูลออกเป็นส่วนย่อย ๆ ให้เหมาะสมกับลักษณะการทำงานของฮาร์ดแวร์ที่ใช้ในเน็ตเวิร์ก เช่น หากชั้นที่ 5 ต้องการส่งข้อมูลที่มีความยาวมากเกินกว่าที่ระบบเครือข่ายจะส่งได้ ชั้นที่ 4 ก็จะทำหน้าที่ตัดข้อมูลออกเป็นส่วนย่อย ๆ แล้วส่งไปให้ผู้รับ ข้อมูลที่ได้รับปลายทางก็จะถูกนำมาต่อกันที่ชั้นที่ 4 ของด้านผู้รับ และส่งให้ชั้นที่ 5 ต่อไป

3.5 ชั้นที่ 3 เน็ตเวิร์กเลเยอร์ (Network Layer)

ทำหน้าที่เชื่อมต่อคอมพิวเตอร์ของด้านรับและด้านส่งเข้าหากันผ่านระบบเครือข่ายพร้อมทั้งเลือกหรือกำหนดเส้นทางที่จะใช้ในการรับส่งข้อมูลระหว่างกัน และส่งผ่านข้อมูลที่ได้รับไปยังอุปกรณ์ในเครือข่ายต่าง ๆ จนกระทั่งถึงปลายทาง ในชั้นที่ 3 นี้ข้อมูลที่รับส่งกันจะอยู่ในรูปแบบ

ของกลุ่มข้อมูลที่เราเรียกว่าแพคเก็ต (Packet) หรือเฟรม (Frame) ข้อมูลชั้นที่ 4,5,6 และ 7 มองเห็นเป็นคำสั่งและไดอะล็อก(Dialog)ต่าง ๆ นั้น จะถูกแปลงและผนึกรวมอยู่ในรูปของแพคเก็ต (Packet) หรือเฟรม (Frame) ที่มีเพียงแอดเดรสของผู้รับ, ผู้ส่ง ลำดับการรับส่งและส่วนของข้อมูลเท่านั้น ตัวเนื้อหาของข้อมูลจะไม่มีผลใด ๆ ในการรับส่งข้อมูลเลย ไม่ว่าข้อมูลในระดับสูงจะเป็นวิดีโอ, ภาพ,เสียง หรือข้อมูลอื่นใดก็ตาม แต่ในชั้นที่ 3 จะมองข้อมูลท้องหมดเป็นแพคเก็ต (Packet)หรือเฟรม (Frame) เท่านั้นหน้าที่อีกประการหนึ่งของชั้นที่ 3 นี้ คือการทำคอลเซตอัพ (Call Setup)หรือเคลียริ่ง (Clearing) หรือยกเลิกการติดต่อเมื่อการรับส่งข้อมูลจบลงแล้ว ในกรณีที่การรับส่งข้อมูลนั้นต้องมีการติดต่อกันก่อน

3.6 ชั้นที่ 2 ดาต้าลิงค์เลเยอร์ (Datalink Layer)

เป็นชั้นที่ทำหน้าที่เชื่อมต่อการรับส่งข้อมูลในระดับฮาร์ดแวร์ โดยเมื่อมีการสั่งให้รับข้อมูลจากในชั้นที่ 3 ลงมา ชั้นที่ 2 จะทำหน้าที่แปลคำสั่งนั้นให้เป็นคำสั่งควบคุมฮาร์ดแวร์ที่ใช้รับส่งข้อมูล ทำการตรวจสอบข้อผิดพลาดในการรับส่งข้อมูลของระดับฮาร์ดแวร์ และแก้ไขข้อผิดพลาดที่ตรวจพบนั้น ข้อมูลที่อยู่ในชั้นที่ 2 นี้ จอยู่ในรูปของเฟรม (Frame) คือกลุ่มของข้อมูลที่มีรูปร่างตามข้อบังคับของฮาร์ดแวร์ที่ใช้ในการรับส่งข้อมูล เช่น ถ้าฮาร์ดแวร์ที่ใช้เป็นอีเธอร์เน็ตแลน (Ethernet LAN) ข้อมูลก็จะมีรูปร่างของเฟรม (Frame) ตามที่ระบุไว้ในมาตรฐานของอีเธอร์เน็ต (Ethernet) หากว่าฮาร์ดแวร์ที่มช้รับส่งข้อมูลเป็นชนิดอื่น เช่น โทคเก้นริงแลน (Token Ring LAN) หรือ ไฟเบอร์ดิสทิบิวท์ดาต้าอินเตอร์เฟซ (Fiber Distributed data Interface :FDDI) รูปร่างของเฟรม (Frame) ที่ใช้ในการรับส่งข้อมูลก็จะเปลี่ยนไปตามมาตรฐานนั้น ๆ

3.7 ชั้นที่ 1 ฟิสิคอลเลเยอร์ Physical Layer

เป็นชั้นล่างสุดของขั้นตอนในการรับส่งข้อมูลของโอเอสไอโมเดล (OSI Model) ซึ่งเป็นชั้นเดียวที่มีการเชื่อมต่อกันทางกายภาพระหว่างคอมพิวเตอร์สองระบบที่ทำการรับส่งข้อมูลกัน ในชั้นที่ 1 นี้ จะกำหนดคุณสมบัติทางกายภาพของฮาร์ดแวร์ที่ใช้เชื่อมต่อระหว่างคอมพิวเตอร์ทั้งสองระบบ เช่น สายที่ใช้รับส่งข้อมูลจะเป็นแบบไหน ข้อต่อหรือปลั๊กที่ใช้ในการรับส่งข้อมูลมีมาตรฐานอย่างไรให้ไฟกี่โวลต์ความเร็วในการรับส่งข้อมูลเป็นเท่าใด สัญญาณที่ใช้รับส่งข้อมูลในสายมีรูปร่างอย่างไร ข้อมูลในชั้นที่ 1 นี้จะมองเห็นเป็นการรับส่งข้อมูลทีละบิตเรียงต่อกันไป โดยไม่มีการพิจารณาเรื่องความหมายของข้อมูลเลย การรับส่งจะส่งข้อมูล “0” หรือ “1” ไปให้คอมพิวเตอร์ด้านรับข้อมูลในระดับฮาร์ดแวร์เท่านั้นหากการรับส่งข้อมูลมีปัญหาเนื่องจากฮาร์ดแวร์

เช่น สายสัญญาณที่ใช้รับส่งขาด อุปกรณ์เสียหาย ก็จะเป็นหน้าที่ของชั้นที่ 1 นี้เช่นกันที่จะตรวจสอบและแจ้งข้อผิดพลาดนั้นให้ชั้นอื่น ๆ ที่อยู่เหนือขึ้นไปทราบ

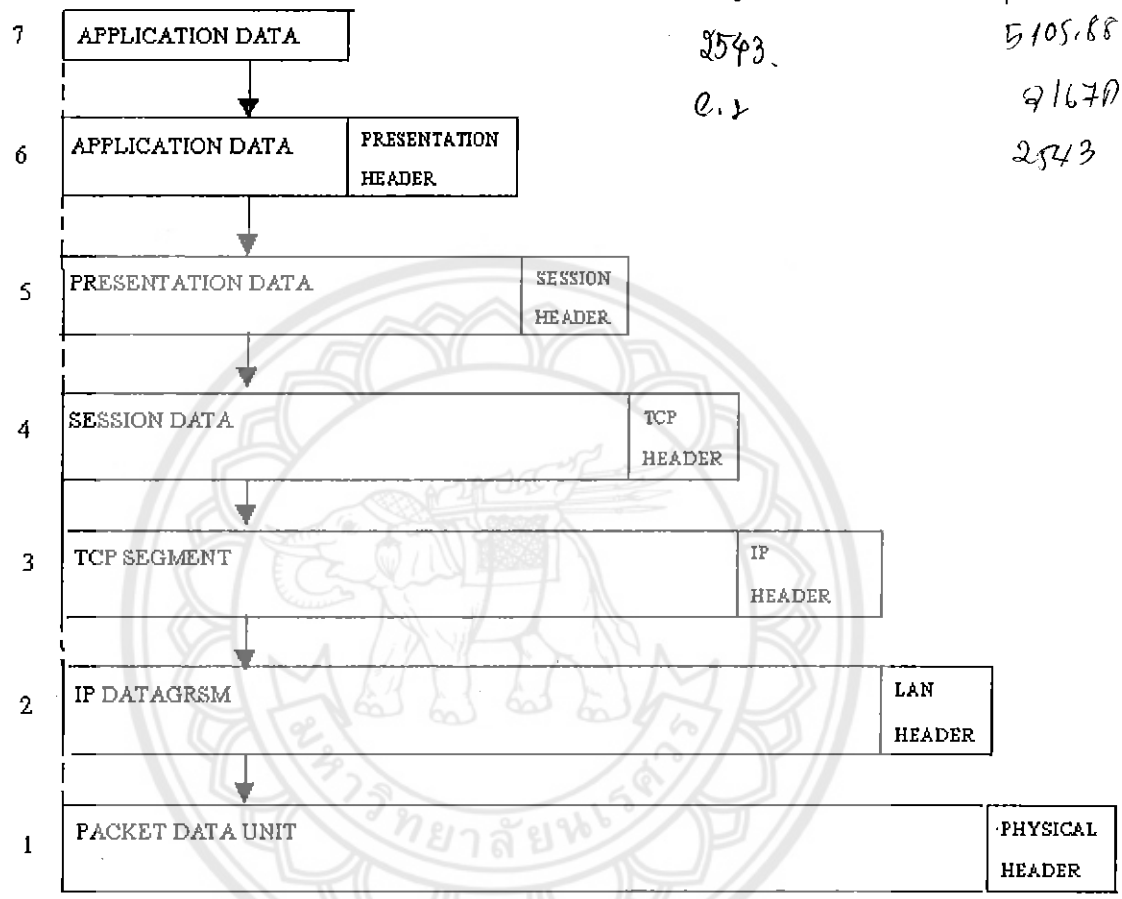
เชื่อมต่อกับผู้ใช้ และแปลคำสั่งต่างๆให้กับคอมพิวเตอร์อย่างถูกต้องตามกฎ	Application Layer	7
แปลงคำสั่งตามกฎที่ได้รับออกเป็นขั้นตอนย่อย ๆ แต่ละขั้นตอน	Presentation Layer	6
ควบคุมจังหวะการรับส่งข้อมูลของคอมพิวเตอร์ทั้งสองด้านให้โต้ตอบกันตามวิธีที่กำหนด(Full/Haft Duplex)	Session Layer	5
เชื่อมต่อการรับส่งข้อมูลจากปลายด้านหนึ่งกับปลายทาง รวมทั้งควบคุมข้อผิดพลาดและตัดข้อมูลออกเป็นส่วนย่อย	Transport Layer	4
ติดต่อกำหนดเส้นทางการรับส่งข้อมูลผ่านเครือข่ายและตรวจสอบ Address ของผู้รับ	Network Layer	3
ควบคุมการรับส่งข้อมูลในระดับฮาร์ดแวร์และตรวจสอบข้อผิดพลาดในการรับส่งข้อมูล	Datalink Layer	2
กำหนดคุณสมบัติของการเชื่อมต่อรับส่งข้อมูลทางฮาร์ดแวร์ ความเร็วในการรับส่ง และการเชื่อมต่อเข้ากับสายรับส่งข้อมูล	Physical Layer	1

รูปที่ 2.11 หน้าที่ของแต่ละชั้นในโอเอสไอโมเดล

(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

ในการรับส่งข้อมูลในโอเอสไอโมเดล นั้น ข้อมูลจากชั้นบนสุด คือชั้นที่ 7 เมื่อถูกส่งลงไปในชั้นถัดลงไป ข้อมูลเดิมก็จะถูกผนึกรวมกับข้อมูลที่ใช้ควบคุมของแต่ละชั้นซ้อน ๆ กันเป็นลำดับเท่ากับจำนวนชั้นที่ผ่านลงไป ตัวอย่างเช่น แอปพลิเคชันดาต้า (Application Data) เมื่อถูกส่งลงไปยังชั้นถัดไปก็จะถูกผนึกด้วยแอปพลิเคชันเฮดเดอร์ (Application Header) และทั้งแอปพลิเคชันเฮดเดอร์ (Application Header) และ แอปพลิเคชันดาต้า (Application Data) จะรวมกันเป็นข้อมูลของชั้นที่อยู่ถัดลงไปอีก ซึ่งชั้นที่อยู่ถัดลงไปอีกทีก็จะผนึกข้อมูลนี้ด้วยเฮดเดอร์ (Header) ของมันเองอีกครั้งหนึ่ง และทั้งเฮดเดอร์ (Header) และข้อมูลเดิมนี้ก็จะกลายเป็นข้อมูลในชั้นถัดลงไปอีกเรื่อย ๆ เป็นเช่นนี้จนกระทั่งถึงชั้นล่างซึ่งเป็นฟิสิคอลเลเยอร์ (Physical Layer) ซึ่งเมื่อข้อมูลถูกส่งไปถึงปลายทาง ข้อมูลที่ได้รับจะถูกแยกเฮดเดอร์ (Header) ที่เพิ่มเข้ามานี้ออกทีละชั้น ซึ่งเป็น

ขบวนการย้อนกลับกับด้านส่ง จนกระทั่งถึงชั้นบนสุด จึงจะเป็นข้อมูลของแอปพลิเคชันดาต้า (Application Data)ให้ผู้รับตามต้องการ ดังแสดงในรูปที่ 2.12

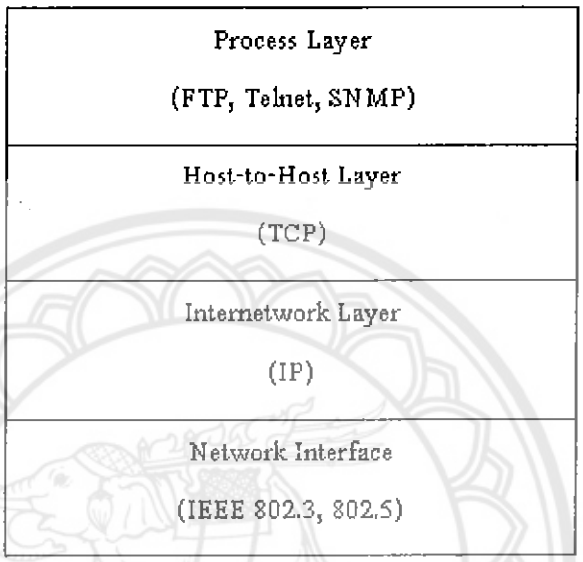


รูปที่ 2.12 การรับส่งข้อมูลแต่ละชั้นของทีซีพี/ไอพี ในโอเอสไอโมเดล
(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณเชียและคณะ)

4. โอเอสไอ กับ ทีซีพี/ไอพี

เนื่องจากโอเอสไอ (OSI) เกิดขึ้นมาหลังจากที่ทีซีพี/ไอพี (TCP/IP หรือ Transmission Control Protocol / Internet Protocol ได้มีการใช้งานกันอย่างแพร่หลายไปแล้ว โดย ทีซีพี/ไอพี ใช้ในเครือข่ายอาร์พาเน็ต (ARPANET) เป็นเครือข่ายแรก ซึ่งต่อมาได้ขยายการเชื่อมต่อไปทั่วโลกเป็นเครือข่ายอินเตอร์เน็ต ทำให้มาตรฐาน ทีซีพี/ไอพี เป็นที่ยอมรับกันอย่างกว้างขวางและการที่ ทีซีพี/ไอพี เป็นโปรโตคอลชนิดที่ใช้ได้ฟรีไม่ต้องจ่ายค่าลิขสิทธิ์ การใช้งาน ทีซีพี/ไอพี ก็ยิ่งมีจำนวนผู้ใช้เพิ่มมากขึ้นไปอีกจนถือเป็นมาตรฐานที่มีผู้ใช้รับส่งข้อมูลมากที่สุดในปัจจุบัน

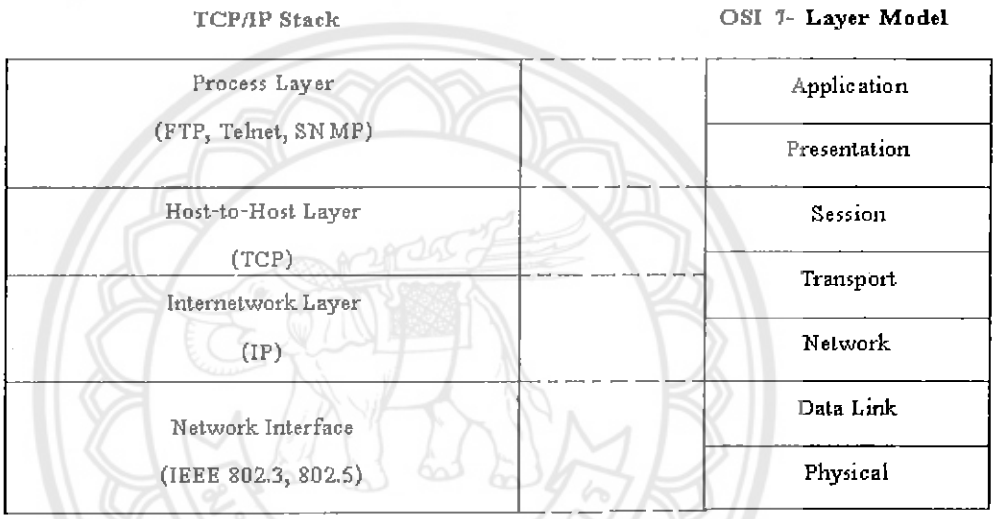
เมื่อ ทีซีพี/ไอพี เป็นมาตรฐานที่เกิดขึ้นก่อนโอเอสไอ โดยทีซีพี/ไอพี จะมีการแบ่งจำนวนขั้นตอนที่ใช้รับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์สองระบบออกเป็น 4 ชั้นเท่านั้น หรือเรียกว่าเป็น ทีซีพี/ไอพี สแตค โดยมีชื่อเรียกแตกต่างกันดังนี้ (ดูรูปที่ 2.13 ประกอบ)



รูปที่ 2.13 โครงสร้างโปรโตคอล ทีซีพี/ไอพี
(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

- ชั้นบนเรียกว่าโปรเซสเลเยอร์ จะเป็นแอปพลิเคชันโปรโตคอล (Application Protocol) ที่ทำหน้าที่เชื่อมต่อกับผู้ใช้และให้บริการต่าง ๆ เช่น FTP,Telnet,SNMP ฯลฯ
- ชั้นถัดมาเรียกว่าโฮสต์ทูโฮสต์เลเยอร์ จะเป็นทีซีพีหรือยูดีพี ที่ทำหน้าที่คล้ายกับชั้นที่ 4 ของ โอเอสไอโมเดล คือควบคุมการรับส่งข้อมูลจากปลายด้านส่งถึงปลายด้านรับข้อมูล และตัดข้อมูลออกเป็นส่วนย่อยให้เหมาะกับเครือข่ายที่ใช้รับส่งข้อมูล รวมทั้งประกอบข้อมูลส่วนย่อย ๆ นี้เข้าด้วยกันเมื่อถึงปลายทาง
- ชั้นถัดลงมาคือ อินเตอร์เน็ตเวิร์กเลเยอร์ (Internetwork Layer) ได้แก่ส่วนของโปรโตคอลไอพี ซึ่งทำหน้าที่คล้ายกับชั้นที่ 3 ของโอเอสไอโมเดล(OSI 7-Layer Model) คือเชื่อมต่อคอมพิวเตอร์เข้ากับระบบเครือข่ายที่อยู่ชั้นล่างลงไปและทำหน้าที่เลือกเส้นทางการรับส่งข้อมูลผ่านอุปกรณ์เครือข่ายต่าง ๆ จนไปถึงผู้รับข้อมูล ในชั้นนี้จะจัดการกับกลุ่มข้อมูลในลักษณะที่เรียกว่าเฟรม (Frame) ในรูปแบบของ ทีซีพี/ไอพี ที่เรารู้จักกันนั่นเอง

- ส่วนชั้นสุดท้ายที่อยู่ล่างสุด เรียกว่าเน็ตเวิร์กอินเตอร์เฟซ (Network Interface) คือชั้นที่ควบคุมฮาร์ดแวร์การรับส่งข้อมูลผ่านเครือข่าย ซึ่งเทียบได้กับชั้นที่ 1 และ 2 ของโอเอสไอโมเดล (OSI 7-Layer Model) ในชั้นนี้จะทำหน้าที่เชื่อมต่อกับฮาร์ดแวร์ และควบคุมการรับส่งข้อมูลในระดับฮาร์ดแวร์ของเครือข่ายซึ่งที่ใช้กันอยู่จะเป็นตามมาตรฐานของไอทริปเปิลอี (IEEE) เช่น IEEE 802.3 จะเป็นการเชื่อมต่อผ่านแลน (LAN) แบบอีเธอร์เน็ต (Ethernet LAN) หรือ IEEE 802.5 จะเป็นการเชื่อมต่อผ่านแลน (LAN) แบบโทคเก้นริง (Token Ring) เป็นต้น



รูปที่ 2.14 โปรโตคอลทีซีพี/ไอพี เมื่อเทียบกับโอเอสไอโมเดล
(ที่มา:เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต สุวัฒน์ ปุณณชัยและคณะ)

ถึงแม้ว่าทีซีพี/ไอพีจะไม่ได้มีการแบ่งชั้นของการสื่อสารข้อมูลตรงตามโอเอสไอโมเดล และไม่ได้เป็นมาตรฐานเดียวกัน แต่โอเอสไอ ก็ออกแบบมาให้เปิดกว้างและเข้ากันได้ดีกับทีซีพี/ไอพี โดยทีซีพี จะเทียบได้กับประมาณชั้นที่ 4 ของโอเอสไอ และไอพี จะเทียบได้กับประมาณชั้นที่ 3 ของโอเอสไอ แม้ว่าจะไม่ลงตัวตรงกันพอดีนัก แต่ก็สามารถเชื่อมต่อทำงานด้วยกันได้ ทำให้มาตรฐานของโอเอสไอ สามารถนำ ทีซีพี/ไอพี มาใช้งานร่วมกันได้เป็นอย่างดี เมื่อเรากลับไปมองมาตรฐานของโอเอสไอโมเดล (OSI 7-Layer Model) ที่เปิดกว้างให้เราเลือกใช้มาตรฐานต่าง ๆ ของแต่ละชั้นมาใช้งานร่วมกันแล้ว จะพบว่าการกำหนดมาตรฐานของโอเอสไอ ได้บรรลุวัตถุประสงค์เป็นอันมาก คือเราสามารถเลือกใช้อุปกรณ์ฮาร์ดแวร์เครือข่าย และโปรแกรมควบคุมในชั้นที่ 1 และ 2 จากบริษัทใดก็ได้มาเชื่อมต่อเข้าด้วยกัน แล้วนำ ทีซีพี/ไอพี ซึ่งมีการใช้งานกันอย่างแพร่หลายมาใช้ในชั้นที่ 4 และ 3 ตามลำดับ ส่วนชั้นที่ 5 ถึงชั้นที่ 7 จะเป็นแอปพลิเคชันที่ต้องการ

2.2 การทำงานแบบไคลเอนต์/เซิร์ฟเวอร์ บนอินเตอร์เน็ต

ไคลเอนต์ ในแง่ของการทำงานแบบไคลเอนต์/เซิร์ฟเวอร์ หมายถึงฝ่ายที่ทำให้เกิดรายการทำงานขึ้น ซึ่งอาจเป็นระบบคอมพิวเตอร์ทั้งระบบ ฮาร์ดแวร์หรือซอฟต์แวร์ที่ใช้โดยคนที่เป็นไคลเอนต์(ผู้ใช้บริการ) โดยหน้าที่หลักของไคลเอนต์คือให้บริการด้านการติดต่อกับผู้ใช้ ช่วยให้ผู้ใช้ที่ต้องการทำงานอย่างใดอย่างหนึ่งระบุความต้องการหรือคำร้องขอ จากนั้นไคลเอนต์ซอฟต์แวร์จะแปลงคำขอให้อยู่ในรูปแบบที่เซิร์เวอร์เข้าใจได้ แล้วจึงส่งไปยังเซิร์ฟเวอร์

เมื่อ เซิร์ฟเวอร์ ได้รับคำร้องขอจากเครือข่าย ก็จะวิเคราะห์ และมักทำการค้นหาหรือปรับข้อมูลในฐานข้อมูล ผลจากการทำงานกับฐานข้อมูลจะถูกส่งกลับไปยังไคลเอนต์ที่ส่งคำขอมาผ่านทางเครือข่าย เมื่อไคลเอนต์ได้รับผลแล้วก็จะแสดงข้อมูลขึ้นบนจอ (หรืออาจเป็นเครื่องพิมพ์เช่นในกรณีของใบเสร็จรับเงิน) เป็นอันเสร็จขั้นตอนการทำงาน

หลักการทำงานของ ไคลเอนต์ /เซิร์ฟเวอร์

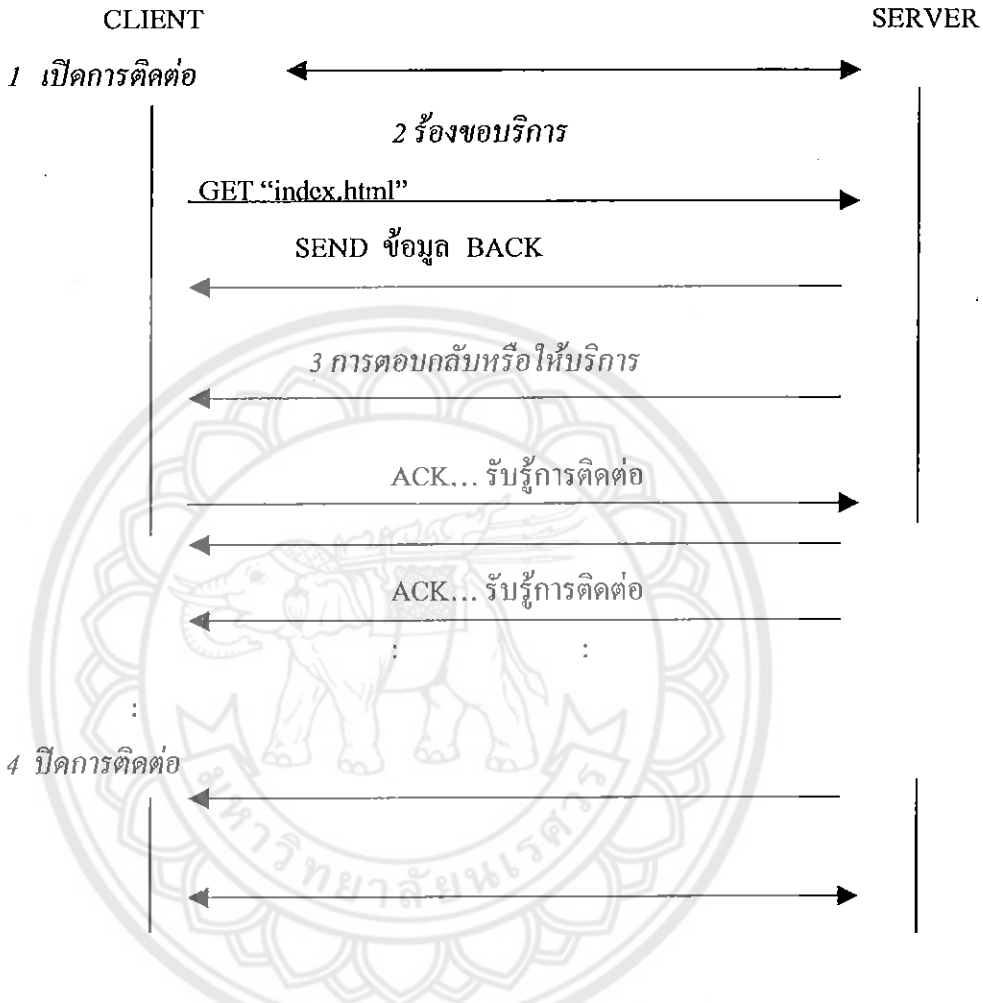
1. ไคลเอนต์ (Client) เป็นผู้ขอใช้บริการ
2. ระบบเครือข่าย (Network)
3. เซิร์ฟเวอร์ (Server) เป็นผู้ให้บริการแก่ไคลเอนต์

โดยมีขั้นตอนการทำงานดังนี้คือ

ขั้นแรก ไคลเอนต์ (จะใช้เว็บเบราเซอร์ เป็น ไคลเอนต์ หลัก ที่จะใช้ร้องขอทรัพยากรไปยังเซิร์ฟเวอร์) จะสร้างการเชื่อมต่อกับ เซิร์ฟเวอร์ ผ่านสิ่งที่เรียกว่า ซ็อกเก็ต (socket) เมื่อ ซ็อกเก็ตทั้งสองฝ่ายเสียบเชื่อมต่อกันได้สำเร็จ ไคลเอนต์จะส่งคำร้องขอข้อมูล (request)ไปยังเซิร์ฟเวอร์จากนั้นเซิร์ฟเวอร์จะไปหาข้อมูลที่ไคลเอนต์ ต้องการ ซึ่งไม่ว่าจะมีหรือไม่มีข้อมูลตามที่ ไคลเอนต์ ร้องขอ เซิร์ฟเวอร์ ก็จะต้องส่งข้อมูลตอบสนอง (response) กลับมายัง ไคลเอนต์ เสมอ สุดท้ายการเชื่อมต่อจะถูกตัดขาด หรือปลดการเชื่อมต่อของ ซ็อกเก็ต ทั้งสองฝ่ายออก

ด้วยการทำงานของ โปรโตคอล HTTP ที่มีการเชื่อมต่อในระยะเพียงสั้น ๆ หรือที่เรียกว่าเป็นโปรโตคอล แบบ connectionless ในลักษณะดังกล่าว ทำให้ช่วงระยะเวลาหนึ่ง ๆ เซิร์ฟเวอร์ ที่ให้บริการ www สามารถรองรับ ไคลเอนต์ ได้จำนวนมากพร้อม ๆ กัน เพราะไม่มีใครทำการเชื่อมต่ออย่างถาวร

แสดงการทำงานของไคลเอนต์/เซิร์ฟเวอร์



รูปที่ 2.15 การทำงานของไคลเอนต์/เซิร์ฟเวอร์

(ที่มา : CGI WEB programming ฉลองชัย จงประเสริฐพร)

สมมุติไฟล์โฮมเพจของ เซิร์ฟเวอร์ หนึ่งถูกกำหนดไว้เป็นชื่อ index.html และมีเนื้อไฟล์ดังนี้

```
<html>
<head><title>Sample Homepage></title><head>
<body background="bkground.gif">
<img src= "under.gif ">
</body>
</html>
```

เมื่อเราสั่งให้เว็บเบราเซอร์ติดต่อไปยัง เซิร์ฟเวอร์ ดังกล่าวนี้ จะต้องค้นหาเซิร์ฟเวอร์ ให้ได้ก่อนเพื่อทำการเชื่อมต่อเมื่อเชื่อมต่อได้แล้ว เว็บเบราเซอร์จะส่งข้อความร้องขอไปยังเซิร์ฟเวอร์ เมื่อเซิร์ฟเวอร์ รับทราบการร้องขอแล้วพบว่าไม่มีการบอกระบุชื่อไฟล์ index.html เมื่อเว็บเบราเซอร์ได้รับไฟล์แล้ว เซิร์ฟเวอร์ จะตัดการเชื่อมต่อทันที ต่อจากนั้น เมื่อเว็บเบราเซอร์เริ่มอ่านและตีความหมายของเท็กในไฟล์index.htmlพบว่ารูปพื้นแบ็กกราวนด์ของเว็บเพจระบุให้ใช้ไฟล์ bkground.gif ดังนั้น ต้องสร้างการเชื่อมต่อไปยัง เซิร์ฟเวอร์ เป็นครั้งที่สอง เพื่อขอไฟล์ bkground.gif จาก เซิร์ฟเวอร์ อีก เมื่อ เซิร์ฟเวอร์ ส่งไฟล์ bkground.gif มาให้เว็บเบราเซอร์แล้ว การเชื่อมต่อก็จะถูกตัดขาดอีกครั้งหนึ่ง จากนั้นเมื่อเว็บเบราเซอร์อ่านพบในบรรทัดต่อมาว่าต้องการแสดงรูปชื่อ under.gif ในเว็บเพจ เว็บเบราเซอร์ก็จะสร้างการเชื่อมต่อเพื่อขอข้อมูลอีก การทำงานของเว็บเพจจะเป็นอย่างนี้เรื่อย ๆ ดังนั้นการเปิดเว็บเพจหนึ่ง ๆ จึงอาจมีการเชื่อมต่อกับ เซิร์ฟเวอร์ เพื่อขอข้อมูลหลายครั้ง

2.3 HTTP (HyperText Transfer Protocol)

HTTP เป็นโปรโตคอลยอดนิยมในอินเตอร์เน็ต โปรโตคอลนี้สร้างขึ้นสำหรับบริการที่เรียกว่า WWW(World Wide Web) ในเครือข่ายอินเตอร์เน็ตโดยเฉพาะ โปรโตคอลนี้จะเป็นตัวกำหนดวิธีการส่งข้อมูลหรือไฟล์ (ส่วนมากจะเรียกรวมว่า ทรัพยากร หรือ resource) ระหว่างเครื่องคอมพิวเตอร์ที่เป็นไคลเอนต์กับเครื่องคอมพิวเตอร์ที่เป็นเซิร์ฟเวอร์รวมถึงกำหนดกฎระเบียบ

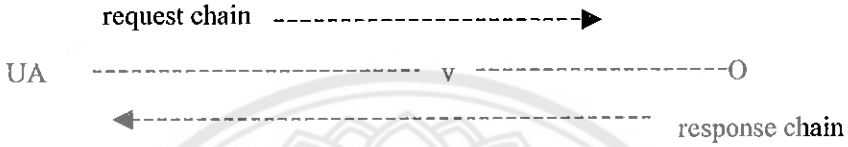
HTTP ถูกพัฒนาโดยนายเบอร์เนอร์ ลี (Berners-Lee) แห่ง CERN ในช่วงปี ค.ศ. 1990-1991 โปรโตคอลนี้ช่วยใช้บริการ WWW ได้รับความนิยมและแพร่หลายมากขึ้นกว่าเดิม เพราะเป็นตัวควบคุมการรับส่งข้อมูลได้ทั้งภาพและเสียงจนเกือบจะเป็นมัลติมีเดีย จากเดิมที่แลกเปลี่ยนได้เพียงข้อความอย่างเดียวรูปแบบจะเป็นแบบ "Connection Oriented" และการทำงานพื้นฐานจะมีรูปแบบเป็นลักษณะแบบ "Transaction Oriented" คือจะอาศัยหลักการง่ายๆ ของ ไคลเอนต์-เซิร์ฟเวอร์ (Client – Server) ในการร้องขอบริการ ซึ่งข้อมูลต่างๆที่เป็นส่วนประกอบของโฮมเพจที่ร้องขอบริการ เช่น ภาพ เสียง และอื่น ๆ จะมีการติดต่อใหม่ เป็นอิสระแก่กัน

เกี่ยวกับ เครือข่ายเวิลด์ไวด์เว็บ (World Wide Web) หรือเรียกสั้นๆ ว่า เว็บ เริ่มด้วยข้อมูลที่ใช้ในระบบเว็บเป็นข้อมูลในลักษณะ interactive Hypermedia กล่าวคือรูปแบบหรือเอกสารที่ใช้งานที่เรียกว่า Hypertext ซึ่งหมายถึง เอกสารที่สามารถ เชื่อมโยงกับเอกสารต่าง ๆ ที่มีความสัมพันธ์กัน โดยภาษาที่ใช้เป็นข้อกำหนดในการสร้างเอกสารรูปแบบนี้คือภาษา HTML (HyperText Markup Language) และวิธีการหรือรายละเอียด ข้อกำหนดในการ รับ-ส่ง ข้อมูลของระบบเว็บ จะอาศัย

โปรโตคอล HTTP (HyperText Transfer Protocol) และตัวรูปแบบข้อมูลจะเรียกว่า Hypermedia เอกสารที่เป็นข้อกำหนดของ HTTP จะเป็นชุดเอกสาร RFC 1945 ซึ่งมีข้อกำหนดดังนี้

1. Overall Operation

การสื่อสารของ HTTP จะเริ่มโดย user agent และส่วนประกอบของการร้องขอไปยังแหล่งข้อมูลบน origin เซิร์ฟเวอร์ ในกรณีตัวอย่าง จะเป็นการร้องขอ โดยการส่งผ่านข้อมูลผ่านทาง user agent (UA) และ origin server (O) โดยตรง โดยแสดงดังรูป



เหตุการณ์ที่เกิดจะสลับซับซ้อนโดยใช้สื่อหนึ่งอย่างหรือมากกว่านั้นซึ่งจะถูกแสดงใน chain การร้องขอ / ตอบกลับ มี 3 รูปแบบ โดยข้อมูลจะเป็นแบบ intermediary ผ่านทาง proxy , gateway และ tunnel

การติดต่อสื่อสารแบบ HTTP โดยทั่วไปจะครอบคลุมถึงเรื่องของการติดต่อแบบ TCP/IP โดยมีการ default port 80 แต่อาจจะมีการใช้ พอร์ต อื่นแทนได้แล้วแต่ผู้ใช้

ซึ่งทั้งหมดที่กล่าวมานี้เป็นเรื่องที่เกี่ยวกับ HTTP ซึ่งเป็นพื้นฐานสำคัญมากในการที่จะติดต่อกับ โปรโตคอลอื่น ๆ บน internet หรือบน network

2. Augmented BNF

ลักษณะพิเศษอีกอย่างหนึ่งที่เกี่ยวกับคำศัพท์ของเอกสารอ้างอิงนี้ ซึ่งเป็นคำบรรยาย ทั้งแบบร้อยแก้วและแบบ Backus-Naur Form (BNF) ซึ่งเหมือนกับการใช้ใน RFC 822 โดย BNF มีรูปแบบดังนี้

- name = definition

เป็นการให้คำนิยาม โดยแบ่งแยกด้วยเครื่องหมาย = (ไม่รวมเครื่องหมาย “<” และ “>”) โดยแบบนี้ใช้กับข้อความเช่น SP , LWS ,HT, CRLF , DIGIT , ALPHA เป็นต้น

- “literal”

จะเป็นเครื่องหมายแสดงข้อความโดยจะปิดหัวและท้ายข้อความ โดยจะยกเว้นข้อความไม่มีความหมาย

- rule1 | rule2

จะใช้เครื่องหมาย | แบ่งข้อความออก โดยจะใช้สำหรับเป็นการเลือก เช่น “ yes | no “ จะเป็นการเลือกว่าจะใช้ yes หรือ no

- (rule1 rule2)

จะเป็นเครื่องหมายวงเล็บปิดข้อความ โดยจะแสดงเหมือนกับมีเพียงข้อความเดียว เช่น "(elem (foo | bar) elem)" จะตัดคำตามลำดับดังนี้ "elem foo elem" และ "elem bar elem"

- *rule

จะเป็นเครื่องหมาย * ก่อนตามด้วยองค์ประกอบโดยจะเป็นการกล่าวซ้ำ โดยรูปแบบเต็มคือ "<n>* <m> element" โดย <n>* จะมาก่อน แล้วตามด้วย <m> โดยค่า default ค่าน้อยสุดคือ 0 และ infinity ดังนั้น "*(element)" จะแสดงถึงจำนวนทั้งหมด รวมถึง 0 ด้วย ตัวอย่างเช่น "1*element" คือการร้องขออย่างน้อยคือ 1 และ "1*2 element" จะเป็นการใช้ 1 หรือ 2

- [rule]

เครื่องหมายสี่เหลี่ยมปิด จะเป็นการแสดงถึงความเหมือนกัน เช่น "[foo bar]" จะเหมือนกับ "*1(foo bar)"

- N rule

จะเป็นการกล่าวซ้ำ เช่น "<n>(element)" เหมือนกับ "<n>*<n> (element)" โดยถ้าเป็นตัวเลข 2 ก็จะมี 2 จำนวน 2 ตัว และ ถ้าเป็นตัวอักษร 3 ตัว ก็จะมีชุดตัวอักษร 3 ชุด

- #rule

เครื่องหมาย "#" มีความหมายเหมือนกับ "*" โดยรูปแบบเต็มคือ "<n>#<m> element" จะแสดงส่วนน้อยสุดคือ <n>และส่วนมากสุดคือ <m> โดยถ้ามีมากกว่า 1 จะแบ่งโดยใช้เครื่องหมาย (",") และอาจจะใช้ linear whitespace (LWS) ตัวอย่างเช่น "(*LWs element *(*LWS "," *LWS element)) สามารถแสดงให้เป็น "1# element"

- ; comment

เครื่องหมาย ; จะเป็นเครื่องหมายปิดท้ายประโยคจบบรรทัด

- implied *LWS

เป็นไวยากรณ์แสดงถึงคำพื้นฐานจะเป็น token หรือ quoted-string

3. HTTP Version

HTTP จะใช้ <major>.<minor> เพื่อบ่งบอกเวอร์ชั่นของโปรโตคอล โดยเวอร์ชั่นโปรโตคอลมีจุดมุ่งหมายเพื่อชี้บอกรูปแบบของข้อความและความสามารถเกี่ยวกับการติดต่อสื่อสารแบบ HTTP การติดต่อสื่อสารหรือมีการเพิ่มค่า field ออกไป จำนวน minor จะถูกเพิ่มขึ้นเมื่อการเปลี่ยนแปลงจะถูกทำที่โปรโตคอลโดยเพิ่มรูปแบบลักษณะซึ่งจะไม่มีการเปลี่ยนข้อความ แต่อาจจะมีข้อความหรือความสามารถเพิ่มขึ้นโดยผู้ส่ง จำนวน major ถูกเพิ่มขึ้นเมื่อรูปแบบของข้อความภายในโปรโตคอลมีการ เปลี่ยนแปลง เวอร์ชั่นของข้อความแบบ HTTP ถูกบ่งบอกโดย field HTTP

เวอร์ชั่น ซึ่งอยู่ในบรรทัดแรกของข้อความ ถ้าเวอร์ชั่นโปรโตคอลไม่ได้ถูกระบุผู้รับต้องเข้ารับโดยข้อความที่อยู่ในรูปแบบ HTTP/0.9

HTTP-Version = "HTTP" "/" 1* DIGIT "." 1* DIGIT

ตัวเลข major และ minor จะมีการเปลี่ยนแปลงโดยแบ่งออกเป็นจำนวนเต็มและบางที่ตัวเลขอาจจะเพิ่มสูงขึ้นมากกว่า 1 หลัก ดังนั้น HTTP/2.4 เป็น เวอร์ชั่น ที่ต่ำกว่า HTTP /2.31 และก็ต่ำกว่า HTTP/1.23 ด้วย การที่หมายเลข 0 นำเป็นข้อเกี่ยวข้องกับผู้รับและไม่ก่อให้เกิดผลด้านผู้ส่งเอกสาร โปรโตคอล HTTP เวอร์ชั่น 0.9 และ 1.0 นี้ การส่งข้อมูลการร้องขอหรือการตอบกลับ ถูกกำหนดโดยลักษณะเฉพาะซึ่งรวมถึง HTTP เวอร์ชั่นของ HTTP/1.0

HTTP/1.0 เซิร์ฟเวอร์ มีความต้องการดังนี้

- การยอมรับรูปแบบของ Request-Line ของ HTTP/0.9 และ HTTP/1.0
- เข้าใจรูปแบบการร้องขอ ของ HTTP/0.9 และ HTTP/1.0
- ตอบกลับข้อความที่เป็นมาตรฐานที่เข้าใจกันของทุก โปรโตคอล

HTTP/1.0 ไคลเอนต์ มีความต้องการดังนี้

- ต้องเข้าใจถึง Status-Line การตอบกลับของ HTTP/1.0
- เข้าใจรูปแบบการตอบกลับของ HTTP/0.9 และ HTTP/1.0

proxy และ gateway แอปพลิเคชัน ต้องคำนึงในเรื่องการร้องขอของครั้งต่อไปซึ่งถูกรับในรูปแบบแตกต่างกันจากเวอร์ชั่น HTTP เดิม เพราะว่า เวอร์ชั่นของโปรโตคอล จะบ่งบองถึงความสามารถของผู้ส่ง proxy/gateway ต้องไม่ส่งข้อความให้กับเวอร์ชั่นที่สูงกว่าของตัวเองถ้าการร้องขอของ เวอร์ชั่น ที่สูงกว่าถูกได้รับการร้องขอแล้ว proxy/gateway ต้องลดระดับหรือตอบกลับไปว่า error การร้องขอกับ เวอร์ชั่น ที่ต่ำกว่าบางที่จะต้องมีการ upgrade ก่อนที่จะส่งตอบกลับไปการตอบสนองการร้องขอของ proxy/gateway ต้องปฏิบัติตามความต้องการของ เซิร์ฟเวอร์

3.1 General Syntax

URIs ใน HTTP สามารถแทนหรือถ่ายทอดไปยัง URI ได้ ขึ้นอยู่กับสิ่งแวดล้อมที่ใช้ โดยมีรูปแบบที่แตกต่างกัน 2 รูปแบบ โดย URIs จะมีรูปแบบชื่อ ตามด้วย เครื่องหมาย colon

URI	=(absoluteURI | relativeURI) ["#" fragment]
absoluteURI	=scheme ":" *(uchar | reserved)
relativeURI	= net_path | abs_path | rel_path
net_path	= "//"net_loc [abs_path]
abs_path	= "/" rel_path
rel_path	= [path] [";" paramd] ["?" query]

path	= fsegment * (“/” segment)
fsegment	= 1*pchar
segment	= *pxhar
params	= param * (“;” param)
param	= * (pchar | “/”)
scheme	= 1 * (ALPHA | DIGIT | “+” | “-“ | ”.”)
net_loc	=*(pchar | “;” | “?”)
query	= * (uchar | reserved)
fragment	= *(uchar | reserved)
pchar	= uchar | “:” | “@” | “&” | “=” | ”+ “
uchar	= unreserved | escape
unreserved	= ALPHA | DIGIT | safe | extra | national
escape	= “%” HEX HEX
reserved	= “;” | “/” | “?” | ”:” | ”@” | ”&” | ”=” | ”+”
extra	= “!” | ”*” | ” ‘ ” | ”(“ | ”)” | ”,”
safe	= “$” | ”-“ | ”_” | ”.”
unsafe	= CTL | SP | <”> | ”#” | ”%” | ”<” | ”>”
national	= <any OCTET excluding ALPHA , DIGIT, reserved extra ,safe and unsafe

คำนิยามของสัญลักษณ์ URL และ sematics ดูที่ RFC 1738 และ RFC 1808 ,BNFเหนือตัว national ขึ้นไปจะไม่ยอมให้มีค่าใน URLs โดยดูจาก RFC1783 เพราะว่า HTTP เซิร์ฟเวอร์จะไม่ถูกจำกัดหรือระบุไว้ในชุดคำสั่งเพราะที่แทนที่ตำแหน่งของ real_path และ HTTP proxy อาจจะได้รับการร้องขอแบบ URLs ที่ไม่มีการนิยามโดย RFC 1738

3.2 http URL

รูปแบบของ http เป็นรูปแบบที่ ใช้ติดต่อกับแหล่งข้อมูลบน เน็ตเวิร์ก ผ่านทาง โปรโตคอล HTTP โดยจะแบ่งแยกรูปแบบของข้อความของ http URLs ดังนี้

http_URL	= “http:” “// “ host [“:” port] [abs_path]
host	= <A legal internet host domain name or IP address
port	= *DIGIT

ถึงแม้ว่าโปรโตคอล HTTP จะเป็นอิสระในด้าน ทรานสปอร์ตเลเยอร์โดย HTTP URL ทั้งหมดจะถูกแบ่งแยกแหล่งทรัพยากรข้อมูลโดย TCP ถ้าไม่มี TCP แหล่งข้อมูลจะถูกระบุโดยรูปแบบของ URI

3.3 Date/Time Formats

HTTP/1.0 จะมีรูปแบบเกี่ยวกับการเก็บค่าและการกำหนดระยะเวลา 3 รูปแบบที่แตกต่างกันโดยข้างล่างนี้เป็นตัวอย่างของการเก็บข้อมูลของ Date/Time

Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822 ,updated by RFC 1123

Sunday , 06-Nov-94 08:49:37 GMT ; RFC 850 obsoleted by RFC1036

Sun Nov 6 08:49:37 1994 ; ANSI C's asctime () format

โดยรูปแบบที่ 1 เป็นที่นิยมมากในมาตราฐานของ อินเตอร์เน็ตและตัวอย่างการกำหนดค่า length โดยอยู่ในส่วนย่อยของคำนิยามใน RFC 1123 ทั้งหมดที่กล่าวมา HTTP /1.0 จะใช้ค่า date/time สำหรับประทับตรา โดยจะใช้เวลาสากลเป็นตัวกำหนด Universal Time (UT) ในเรื่องของ GMT ซึ่งอยู่ภายใน โดยจะชี้บอกในรูปแบบ 1 และรูปแบบ 2 โดยรูปแบบของเวลา จะมีดังนี้

```
HTTP-date       = rfc1123-date | rfc850-date | asctime-date
rfc1123-date    = wkday "," SP date1 SP time SP "GMT"
rfc850          = weekday "," SP date2 SP time SP "GMT"
asctime-date    = wkday SP date3 SP time SP 4DIGIT
date1           = 2DIGIT SP month SP 4DIGIT
                  ; day month year (e.g. ,02 Jun 1982)
date2           = 2DIGIT "-" month "-" 2DIGIT
                  ; day-month-year (e.g., 02-Jun-82)
date3           = month SP ( 2DIGIT | (SP 1 DIGIT))
                  ;month day (e.g.,Jun 2)
time            = 2DIGIT ":" 2 DIGIT ":" 2DIGIT
                  ;00:00:00 – 23:59:59
wkday           = "Mon" | "Tue" | "Wed" | "Thu" | "Fri" | "Sat" | "Sun"
weekday         ="Monday" | "Tuesday" | "Wednesday"
                  | "Thursday" | "Friday" | "Saturday" | "Sunday"
month           ="Jun" | "Feb" | "Mar" | "Apr" | "May" | "Jun" | "Jul"
                  | "Aug" | "Sep" | "Oct" | "Nov" | "Dec"
```

3.4 Media Types

HTTP ใช้ internet Media Types ในส่วน Content – Type ซึ่งอยู่ในส่วนเฮดเดอร์

Media type = type "/" subtype *

Type = token

Subtype = token

Parameter จะตาม type/subtype ในฟอร์มของ attribute/value

Parameter = attribute "=" value

Attribute = token

Value = token | quoted-string

3.5 Product Tokens

Product token ถูกใช้ในการติดต่อของ แอปพลิเคชัน เพื่อเป็นการระบุตัวมันเองผ่านทาง simple product token โดยการใช้เครื่องหมาย / และเวอร์ชั่น การใช้ฟิลด์ product token จะใช้ subproduct ซึ่งเป็นฟอร์มส่วนต่าง ๆ ที่สำคัญของ แอปพลิเคชัน เพื่อทำการ list ออกมา โดยจะแบ่งแยกจากกันด้วยช่องว่าง เพื่อให้สะดวกขึ้น products จะถูก list ตามลำดับความสำคัญเป็นการระบุแอปพลิเคชัน นั่นเอง

Product = token ["/" product-version]

Product-version = token

ตัวอย่างเช่น

User-Agent : CERN-LineMode/2.15 libwww/2.17b3

Server : Apache/0.8.4

4. Message Type

HTTP message จะประกอบด้วย ขอความร้องขอ จากไคลเอนต์ส่งไปยังเซิร์ฟเวอร์และตอบสนองจากเซิร์ฟเวอร์ส่งไปยังไคลเอนต์

HTTP-message = simple-request ;HTTP/0.9 message

Simple-response

Full-request ;HTTp/1.0 message

Full-response

Full-request และ Full-reponse ใช้เป็น message แบบทั่วไปในรูปแบบของ RFC 822 สำหรับการ transfer entities message ทั้ง 2 อย่างนี้ จะประกอบด้วย เฮดเดอร์ และ entity body โดยทั้งสองส่วนจะแยกกันด้วยบรรทัดว่าง

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

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

simple-request และ simple-response ไม่อนุญาตให้ใช้ข้อมูลในส่วน เฮดเดอร์ และถูกจำกัดสำหรับ single request method (GET)

simple-request = "GET" SP Request-URI CRLF

simple-response = [Entity-Body]

การใช้รูปแบบ simple-request ถูกห้ามไม่ให้ใช้เนื่องจากมันป้องกันเซิร์ฟเวอร์จากการระบุ media type ของ entity ที่ส่งกลับมา

4.1 Message Headers

ในส่วนเฮดเดอร์ฟิลด์ของ HTTP ประกอบไปด้วย Generl-Header, Request-Header, Response-Header และ Entity-Header fields ซึ่งเป็นรูปแบบทั่วไป เช่นเดียวกับรายละเอียดที่ให้ไว้ในส่วนของ 3.1 ของ RFC 822 แต่ละส่วนของเฮดเดอร์ จะประกอบด้วยชื่อ ตามด้วยเครื่องหมาย : ช่องว่าง (a single space (SP)) ,character และ field value

HTTP-header = field-name ":" [field-value]CRLF

Field-name = token

Field-value = *(field-content |LWS)

Field –content = *<TEXT หรือ combination ของ token,tspecials และ quotedstring>

4.2 General Header Fields

มี Header field ที่ใช้ได้โดยทั่วไปสำหรับ request และ response message แต่ไม่สามารถใช้ได้กับ entity ที่กำลังถ่ายโอนอยู่ เฮดเดอร์ นี้สามารถใช้ได้กับ message ที่กำลัง transmit เท่านั้น

General-Header = Date | Pragma

5. Request

เมื่อ ไคลเอนต์ เชื่อมต่อกับ เซิร์ฟเวอร์ สำเร็จแล้ว ไคลเอนต์ จะเป็นฝ่ายเริ่มเปิดการพูดคุยด้วยการส่งข้อมูลไปยัง เซิร์ฟเวอร์ เพื่อบอกการร้องขอข้อมูล การร้องขอไปยัง เซิร์ฟเวอร์ สามารถทำได้หลายแบบ โดยรูปแบบจะเขียนข้อความในบรรทัดแรกสุดของ HTTP Header ซึ่งประกอบด้วย 3 ส่วน คือ วิธีการร้องขอ (method) , ไดเร็กทอรี่กับชื่อไฟล์ที่ร้องขอและเวอร์ชั่นของ HTTP ที่ผู้ร้องข้อใช้อยู่และแต่ละส่วนของข้อความจะถูกคั่นด้วยช่องว่าง

<u>รูปแบบ</u>

Method /path/file HTTP/x.x

- Method บอกว่าใช้ วิธีการร้องขอ แบบใดจาก เซิร์ฟเวอร์
- /path/file ไดเร็กทอรี่และชื่อไฟล์ที่ต้องการจาก เซิร์ฟเวอร์
- HTTP/x.x x.x คือเวอร์ชั่นของ HTTP บอกให้ เซิร์ฟเวอร์ ทราบว่าทาง ไคลเอนต์ ใช้ HTTP เวอร์ชั่นไหนอยู่

สำหรับ HTTP จะมีคำสั่งหลักๆในการจัดการดังนี้ OPTION ,HEAD, PUT, DELETE, TRACE, GET และ POSTแต่คำสั่งหลัก ๆ สำหรับผู้ขอใช้บริการ มักใช้บ่อยคือ GET, POST และ HEAD โดยรายละเอียดการใช้คำสั่ง สำหรับการร้องขอนั้น จะมีการระบุรายละเอียดไว้ในตัวของโปรโตคอล HTTP

โดยส่วนใหญ่การส่งข้อมูลจากฟอร์มในเว็บเพจไปประมวลผลใน เซิร์ฟเวอร์ จะใช้ Method นี้มากที่สุด ข้อมูลจากเว็บเบราเซอร์ ที่จะส่งไปให้ เซิร์ฟเวอร์ จะถูกเข้ารหัส (URL-encode) ก่อนเสมอ เพื่อให้ เซิร์ฟเวอร์ ตีความหมายของข้อมูลที่ร้องขอมาได้อย่างถูกต้อง

แสดงตัวอย่างการร้องขอบริการของ HTTP

Get / index.html HTTP/1.0 <CR><LF>

Connection: Keep-Alive <CR><LF>

User-Agent: Mozilla/2.01(Win95;1) <CR><LF>

Pragma: no-cache <CR><LF>

Host: 161.246.10.21 <CR><LF>

Accept: Image/gif , image/jpeg , image/pjpeg,*/* <CR><LF> <CR><LF>

โดยการแยกรายการต่าง ๆ ของรายละเอียดด้วย “<CR><LF> “ซึ่งคือ “Carriage Return” และ “Line Fees” ตามลำดับในกรณีการโปรแกรมนี้คือค่า “\n” (C/C++,Perl) หรือ “New Line” ใช้ในการแยก รายละเอียดของเฮดเดอร์ ของผลลัพธ์นั้น ๆ โดยใช้จำนวนหนึ่งชุด และเมื่อจบส่วนเฮดเดอร์ของผลลัพธ์จะใช้ “<CR><LF><CR><LF>”

การร้องขอบริการจะมีความเกี่ยวข้องกับการระบุถึงสถานที่ที่ต้องการใช้บริการและรายละเอียดที่อยู่ของข้อมูลต่าง ๆ ตามหลักการของ URL(Uniform Resource Locators) ซึ่ง URL คือส่วนระบุเพิ่มเติมในการเข้าถึงข้อมูลโดยสามารถใช้วิธีการใช้งาน หรือโปรโตคอลได้หลายวิธี

จากการร้องขอบริการข้างต้นจะเป็นการเรียกขอโฮมเพจ ที่ชื่อ "index.html" โดยตรงนี้อาจจะระบุแบบ ส่วนขยาย URL ได้ ส่วนสำคัญถัดมาของการร้องขอบริการ คือส่วนของ User Agent เป็นการระบุรายละเอียดเกี่ยวกับ สถานะภาพของไคลเอนต์ ส่วนต่อมาคือโฮสต์ (Host) ซึ่งจะเป็นไซต์ปลายทางและรายละเอียดนี้จะสัมพันธ์กับส่วนเพิ่มเติมของ URL

รายละเอียดในการเข้ารหัส URL-encode

1. แปลงตัวอักษรหรือเครื่องหมายบางตัวในอยู่ในรูป %xx โดยที่ xx จะเป็นค่าแอสกี้ของตัวอักษรนั้น ตัวอักษรที่มีการแปลงคือ %,&,',+ และ ? เพราะเป็นเครื่องหมายที่ใช้เป็นตัวแยกข้อมูลที่จะส่งไปให้ เซิร์ฟเวอร์

2. เปลี่ยนช่องว่างทุกตัวเป็นเครื่องหมายบวก

3.ระบุชื่อตัวแปร และค่าตัวแปรโดยคั่นกลางด้วยเครื่องหมาย = และคั่นระหว่างตัวแปรด้วยเครื่องหมาย & เช่น

Number= 152.528813&frame=zorkia&message=Just+Do+It

ส่วนประกอบ request message ที่ส่งจากไคลเอนต์ไปยังเซิร์ฟเวอร์ คือ ภายในบรรทัดแรกของ message นี้ จะมี method และ เวอร์ชั่นของโปรโตคอล ที่ใช้รูปแบบของ HTTP request มี 2 แบบ ดังนี้

Request = simple-request | Full request

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

Full-request = Request-Line

*(General-Header | Request-Header | Entity-Header)

CRLF

[Entity-Body]

6. Request-Line

Request เริ่มด้วย method token ตามด้วย request-URI , เวอร์ชั่นของโปรโตคอล ที่ใช้ และจบด้วย CRLF แต่ละส่วนจะแยกกันด้วย SP character ไม่มี CR หรือ LF ตามหลัง ยกเว้นจะมี CRLF ในลำดับสุดท้าย

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

NOTE! ข้อแตกต่างระหว่าง simple-request และ request-line ของ full-request คือ การแสดงในส่วนเวอร์ชั่นของ HTTP และความสามารถในการใช้ method อื่น ๆ นอกจาก GET

6.1 Method

Method token เป็นส่วนแสดงว่าในการ request ขอทรัพยากรนั้นใช้ Method อะไร

Method = "GET" |"HEAD" |"POST"
| extension-method

extension-method = token

6.2 Request-URI

URI คือUniform Resource Idertifier ใช้เพื่อระบุว่าต้องการร้องขอทรัพยากรจากที่ไหน

Request-URI = absoluteURI | abs_path

ตัวอย่าง Request-Line

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

6.3 Request Header Fields

request header fields ใช้ในการให้ข้อมูลเพิ่มเติมจากไคลเอนต์หรือข้อมูลของไคลเอนต์เองส่งไปยังเซิร์ฟเวอร์ ในส่วนนี้จะทำหน้าที่เหมือนกับตัวปรับปรุงหรือเปลี่ยนแปลงคำร้องขอ

Request-Header = Authorization
| From | If-Modified-Since | Referer | User-Agent

7. Response

ในการตอบกลับการให้บริการของHTTPนั้นจะมีรูปแบบในการใช้งานเหมือนโปรโตคอล อื่น ๆ ตามหลักการของไคลเอนต์-เซิร์ฟเวอร์ ทั่วไป คือ จะมีการส่งค่ากลับ นำด้วยหมายเลขซึ่งเรียกว่า "Response Tags Number หรือ Status Code" และจะตามด้วยรายละเอียดข้อความซึ่งอธิบาย จากนั้นส่วนท้ายสุดที่อาจจะมีส่งตามคือตัวของข้อมูลจริงๆ ในกรณีมีการขอข้อมูล เช่น จากการร้องขอ ข้างต้นเราจะได้รับข้อมูล ที่เว็บเซิร์ฟเวอร์ หรือเซิร์ฟเวอร์ส่งกลับมา ไม่ว่าจะมีข้อมูลหรือไม่มีข้อมูลที่ไคลเอนต์ร้องขอเข้ามาก็ตามเซิร์ฟเวอร์จะต้องส่งข้อความตอบสนองกลับไปให้ ไคลเอนต์รับทราบเสมอ ในส่วนของกรณีตอบสนอง ข้อความบรรทัดแรกจะเรียกว่า บรรทัดสถานะ (Status Line) โครงสร้างบรรทัดนี้ มี 3 ส่วน แต่ละส่วนคั่นด้วยช่องว่าง ดังต่อไปนี้

HTTP/x.x xxx Description

- HTTP/x.x x.x คือเวอร์ชั่นของ HTTP เพื่อบอกแก่ไคลเอนต์ว่าทางเซิร์ฟเวอร์เวอร์ชั่นไหนอยู่
- xxx ตัวเลข 3 หลัก เป็นรหัสตอบสนอง
- Description ข้อความอธิบายรหัสตอบสนอง

เมื่อเซิร์ฟเวอร์ได้รับคำร้องขอและได้ทำการแปลความหมายแล้วเซิร์ฟเวอร์ต้องมีการตอบสนองกลับไปยังไคลเอนต์ โดยอยู่ในรูปแบบของ HTTP response message

Response = simple-response | full-response

Simple-response = [Entity-Body]

Full-response = status-line
*(General-Header | Response-Header | Entity-Header)
CRLF
[Entity-Body]

7.1 Status-line

บรรทัดแรกของ Full-response message คือ status-line ซึ่งประกอบด้วย เวอร์ชั่น ของโปรโตคอล ที่ใช้ ตามด้วยตัวเลขซึ่งเป็นรหัสแสดงสถานะ และส่วนแสดงเหตุผล

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

"HTTP/" 1*DIGIT "." 1*DIGIT SP 3DIGIT SP

เช่น "HTTP/1.0 200"

7.2 Status Code and Reason Phras

รหัสแสดงสถานะเป็นตัวเลข 3 ตัว ซึ่งแสดงผลของการ request และมี reason phrase เป็นคำอธิบายสั้น ๆ ของรหัสสถานะ

โปรโตคอล HTTP ได้กำหนดรหัสแสดงสถานะการทำงานของโปรโตคอลไว้ โดยแบ่งกลุ่มของรหัสสถานะออกเป็น 5 กลุ่ม คือ

ตารางที่ 2.2 การแบ่งสถานะการทำงานของโปรโตคอล
(ที่มา : CGI WEB programming ฉลองชัย จงประเสริฐพร)

รหัสสถานะ	ประเภท	รายละเอียด
1XX	Information	เป็นรหัสสถานะกลุ่มที่เปิดให้โปรแกรมประยุกต์ต่าง ๆ กำหนดใช้งานได้เอง
2XX	Successful	กลุ่มรหัสที่แสดงว่าการทำงานสำเร็จ
3XX	Redirection	กลุ่มรหัสนี้จะใช้ภายในโปรโตคอล HTTP เอง โดยเป็นการทำงานที่ต่อเนื่องมาจากโปรเซสก่อนหน้านี้ซึ่งไคลเอนต์เป็นผู้สั่งงาน
4XX	Client Error	ใช้แสดงปัญหาที่เกิดขึ้นกับไคลเอนต์
5XX	Server Error	ใช้แสดงปัญหาที่เกิดขึ้นกับเซิร์ฟเวอร์

7.3 Response Header Fields

เป็นข้อมูลเพิ่มเติมที่เซิร์ฟเวอร์ตอบสนองกลับมายังไคลเอนต์ ซึ่งไม่สามารถเขียนในบรรทัดสถานะได้(status-line)ในส่วนเฮดเดอร์จะมีข้อมูลเกี่ยวกับเซิร์ฟเวอร์และการเข้าถึงทรัพยากรซึ่งระบุโดย request-URI

Reponse-Header = Location | server | WWW-Authenticate

8. Entity

Entity ประกอบด้วย Entity-Header field และ Entity body

8.1 Entity Header Fields

Entity-Header fields จะกำหนด metainformation เกี่ยวกับ Entity-Body หรือถ้าไม่มี body ก็จะแสดงเกี่ยวกับทรัพยากร ซึ่งจะระบุโดยคำร้องขอ

Entity-Header = Allow | Content-encoding | Content-Length
| Content-Type | Expires | Last-Modified
| extension-header

extension-header = HTTP-header

8.2 Entity Body

Entity Body ส่งด้วย HTTP request หรือ response ซึ่งอยู่ในรูปแบบและการเข้ารหัสโดย Entity Header fields

Entity-Body = *OCTET

9. Method Definitions

เซตของ Method โดยทั่วไปสำหรับ HTTP 1.0 มีดังนี้

9.1 GET

รูปแบบ GET /path/file HTTP/x.x เช่น สมมุติถ้าเราป้อน URL ให้กับเว็บเบราเซอร์เป็น http://nu.ac.th/news/current.html เว็บเบราเซอร์ต้องสร้างการเชื่อมต่อไปยัง www.nu.ac.th ก่อน แล้วสร้างข้อความร้องขอส่งไปยัง เซิร์ฟเวอร์ สมมุติเว็บเบราเซอร์สนับสนุน HTTP 1.0 ดังนั้นโครงสร้างของข้อความร้องขอ จะเป็นดังรูป

GET /news/current.html HTTP/1.0
User-Agent ; Mozilla /4.03 (en) (Win95;|) Accept : image / gif , image / jpeg, */ * Accept – Language : en Accept –Charset : iso –8859 –1 , * , utf –8 Cache –Control L max – age = 259200
บรรทัดว่าง

รูปที่ 2.16 โครงสร้างของข้อความร้องขอของไคลเอนต์

(ที่มา : แกะรอย CGI ทรงเกรียรติ ภาวดี)

ข้อความร้องขอไปยังเซิร์ฟเวอร์ตีความหมายจากเฮดเดอร์ได้ว่าเว็บเบราเซอร์ร้องขอเปิดไฟล์ current.html ในไดเร็กทอรี่/news จากโดเมนเนม nu.ac.th และเว็บเบราเซอร์ใช้ โปรโตคอล HTTP เวอร์ชั่น 1.0 สังเกตเห็นว่า นอกจากคำร้องขอที่อยู่บรรทัดแรกแล้ว ยังมีข้อมูลอื่น ๆ ที่เว็บเบราเซอร์ส่งไปให้ เซิร์ฟเวอร์ ด้วย นั่นคือ เฮดเดอร์ จะเป็นตัวบอกรายละเอียดจากผู้ส่งว่ามีอะไรบ้าง ข้อความร้องขอถูกส่งมาจากใคร เป็นเบราเซอร์เวอร์ชั่นไหน ค่ายไหน

9.2 HEAD

การร้องขอข้อมูลด้วย Method HEAD จะคล้ายกับ Method GET คือใช้ร้องขอเพื่อถาม เซิร์ฟเวอร์ ให้ เซิร์ฟเวอร์ ส่งรายละเอียดของไฟล์ที่ ไคลเอนต์ ต้องการกลับมา (Header) เพียงแต่ Method นี้จะไม่มีการส่งบล็อกข้อมูลไปกับข้อความร้องขอเหมือนกับ Method GET

รูปแบบ HEAD /path/file HTTP/x.x

สมมุติว่า ส่งข้อความร้องขอไปยัง เซิร์ฟเวอร์ nu.ac.th ด้วย Method HEAD เพื่อถามว่ามีไฟล์ index.html หรือไม่ โดยระบุข้อความร้องขอในบรรทัดแรกเป็น HEAD/index.html HTTP/1.0 และสมมุติว่ามีไฟล์อยู่จริง และมีขนาด 1,938 Byte ข้อความตอบกลับจาก เซิร์ฟเวอร์ ก็จะได้ ดังรูป

HTTP/1.0 200 OK
Date : Fri , 18 Dec 1998 21:07:46 GMT Last – Modified : Thu , 06 Nov 1997 18:20:06 GMT Content – Type : text/html Content – Length :1938
บรรทัดว่าง

รูปที่ 2.17 ข้อความตอบกลับจากเซิร์ฟเวอร์
(ที่มา : แกะรอย CGI ทรงเกียรติ ภาวดี)

ข้อความตอบสนองที่ได้จากการร้องขอด้วย Method HEAD คำสั่งนี้จะทำงานคล้ายกับคำสั่ง GET แต่เว็บเซิร์ฟเวอร์จะส่งข้อมูลกลับมาให้เฉพาะรายละเอียดของ metadata หรือข้อมูลในส่วนเฮดเดอร์เท่านั้น ส่วนข้อมูลที่เป็น HTML จะไม่ถูกส่งมาด้วย ซึ่งคำสั่ง HEAD นี้ จะใช้เพื่อทดสอบว่าข้อมูลตาม URL นั้น ๆ มีการเปลี่ยนแปลงหรือไม่เท่านั้น

9.3 POST

การร้องขอข้อมูลด้วย Method POST จะใช้ในกรณีที่ต้องการส่งข้อมูลจาก ไคลเอนต์ ไปยัง เซิร์ฟเวอร์ ให้รับไปทำงาน

รูปแบบ POST /path/file HTTP/x.x

- ข้อมูลที่จะส่งไปให้ เซิร์ฟเวอร์ จะอยู่ภายในบล็อกข้อมูล และต้องมี Content Type และ Content Length เพิ่มเข้าไปในเฮดเดอร์ของข้อความร้องขอด้วย

- /path/file คือชื่อโปรแกรม CGI ใน เซิร์ฟเวอร์ ที่จะทำหน้าที่รับข้อมูลไปประมวลผล
- ข้อความตอบสนองที่ เซิร์ฟเวอร์จะส่งกลับไปให้ ไคลเอนต์ จะได้มาจากการทำงานของโปรแกรมใน เซิร์ฟเวอร์

เป็นคำสั่งที่ตรงข้ามกับคำสั่ง GET และ HEAD โดยทำหน้าที่ส่งข้อมูลจากไคลเอนต์ไปยังเซิร์ฟเวอร์ แต่โดยปกติแล้วการส่งข้อมูลจากไคลเอนต์ไปยังเว็บเซิร์ฟเวอร์ นั้นจะไม่ค่อยมีใช้งานนอกจากในกรณีที่ HTMLทำงานในลักษณะที่ให้ผู้ใช้กรอกข้อมูลลงตามแบบฟอร์ม(เช่นรายละเอียดส่วนตัวของผู้ใช้งาน) และส่งข้อมูลนี้กลับมาเก็บไว้ที่เว็บเซิร์ฟเวอร์

10. Status Code Definition

10.1 Information 1XX

รหัสสถานะในกลุ่มนี้บ่งชี้ถึงข้อกำหนดของการตอบสนอง HTTP/1.0 ไม่กำหนดรหัสสถานะ 1XX และไม่สามารถตอบสนองกับ HTTP/1.0 request ได้ แต่อย่างไรก็ตาม มันก็มีประโยชน์อย่างยิ่งกับการทดลอง Application ซึ่งอยู่นอกเหนือ scope ของการกำหนดนี้

100 Contine

101 Switching

10.2 Successful 2XX

200 OK

201 Created

202 Accepted

203 Non –Antharitstive information

204 No Content

205 Reset Content

206 Partial Content

10.3 Redirection 3XX

300 Multiple Permanently

301 Moved Permanently

302 Moved Temporarily

303 See Other

304 Not Modified

305 Use Proxy

10.4 Client Error 4xx

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

10.5 Server Error 5xx

500 Internal Server Error

501 Not Implemented

502 Bad Gateway

503 Service Unavailable

504 Gateway Time – out

505 HTTP Version not supported

11. เฮดเดอร์

เฮดเดอร์เป็นส่วนที่ใช้บอกรายละเอียดต่างๆของข้อมูลทั้งการร้องขอและการตอบสนอง โดยมีรูปแบบเขียนดังนี้

Header-name : value แล้วปิดท้ายด้วยเครื่องหมายลงบรรทัดใหม่ (CR และ LF)รายการเฮดเดอร์ในกรณีตอบสนองที่ เซิร์ฟเวอร์ จะต้องบอกรายละเอียดของข้อมูลไปยัง ไคลเอนต์

ซึ่งมีสาระสำคัญอยู่สองข้อคือ อย่างแรกเรื่องของการกำหนดรายละเอียดของเฮดเดอร์จะใช้ "\n" จำนวนหนึ่งชุดในการแยกรายละเอียดของเฮดเดอร์ และใช้ "\n\n" หรือ"\n"จำนวนสองชุดสำหรับแยกรายละเอียดเฮดเดอร์กับส่วนของข้อมูล และข้อสองคือรายละเอียดเกี่ยวกับ เฮดเดอร์และแต่ละส่วนสำดับก่อนหลังไม่มีความสำคัญในเรื่องของบรรทัดกล่าวคือในส่วนของเฮดเดอร์นั้นจะแจ้งรายละเอียดใดมาก่อนหรือหลังก็ได้ไม่มีผลใด ๆรายละเอียดเฮดเดอร์ ของ โปรโตคอล HTTP

ตารางที่ 2.3 รายละเอียดเฮดเดอร์ ของ โปรโตคอล HTTP
(ที่มา : CGI WEB programming ฉลองชัย จงประเสริฐพร)

เฮดเดอร์ (Header)	รายละเอียด (Description)
Content-length	บอกขนาดของข้อมูลที่ส่งมาด้วยว่ามีขนาดเท่าไรหน่วยเป็นไบต์
Content-type	บอกชนิดของข้อมูลที่ส่งมาว่าเป็นแบบใด
Expires	บอกวันและเวลาสำหรับระบุว่าเอกสารจะหมดอายุเมื่อไร
Location	บอกให้เว็บเซิร์ฟเวอร์ทำการเปลี่ยนทิศทางการส่งผลลัพธ์
Pragma	จะเป็นการกำหนดเรื่องของแคชว่าจะใช้หรือไม่ใช้กับข้อมูลที่ส่งมานั้น ๆ
Status	บอกสถานะของการขอใช้บริการ
Refresh	เป็นการระบุให้ไคลเอนต์ทำการ รีโลด *ข้อมูลพิเศษ* ใหม่
Set-Cookie	เป็นการระบุให้ไคลเอนต์เก็บรายละเอียดเกี่ยวกับ *ข้อมูลพิเศษ*

หัวข้อต่อไปจะกล่าวถึงโครงสร้างและสิ่งที่เกี่ยวข้องกับเฮดเดอร์ใน โปรโตคอล HTTP/1.0 ที่ใช้อ้างอิงในการสื่อสารกันในระบบเครือข่ายแบบ WWW ว่าใครเป็นฝ่ายรับ หรือเป็นฝ่ายส่งข้อความระหว่าง เซิร์ฟเวอร์ กับ ไคลเอนต์

รูปแบบของเฮดเดอร์ (Header Sntax)

11.1 Allow

เป็นเฮดเดอร์ที่ใช้แสดงรายการเมธอดที่รองรับทรัพยากรตาม Request-URI แจ้งให้ทราบถึงเมธอดอะไรบ้างที่เข้าถึงข้อมูลชนิดนั้นๆ ได้ เฮดเดอร์นี้จะไม่ยอมให้มีการเข้าถึงข้อมูลด้วยเมธอด POST และยังขึ้นอยู่กับว่า เซิร์ฟเวอร์ ยอมให้มีการเข้าถึงข้อมูลมากน้อยเพียงใดอีกด้วย แต่ก็ไม่ได้หมายความว่า เซิร์ฟเวอร์ รองรับเมธอดได้เท่ากับที่ระบุในเมธอดนี้

รูปแบบ Allow = "Allow" ":" #method

ตัวอย่าง Allow : GET , HEAD

11.2 Authorization

โปรแกรม(หรือ ไคลเอนต์) ทำการร้องขอตามการขอการรับรองจาก เซิร์ฟเวอร์ หลังจากที่ถูกปฏิเสธการทำงาน (ไคลเอนต์ ที่ไม่ได้รับอนุญาต ซึ่งได้รับรหัสตอบสนองกลับ เป็น 401 (Unauthorized)) ทำโดยเพิ่มเฮดเดอร์นี้เข้าไปในข้อความร้องขอ เมื่อ เซิร์ฟเวอร์ ได้รับคำร้องขอนี้แล้วจะสั่งการตอบสนองกลับมา

รูปแบบ Authorization Authorization = "Authorization" ":" credentials

Credentials คือ ข้อความแสดงการรับรองจาก ไคลเอนต์ ที่ระบุขอบเขตการร้องขอข้อมูล

11.3 Content-Encoding

เฮดเดอร์นี้จะใช้ในการเปลี่ยนแปลงชนิดของข้อมูลที่มีการรบีบอัดซึ่งเป็นลักษณะการแบ่งกลุ่มของข้อมูลที่มาจาก Request-URI ใช้ร่วมกับเฮดเดอร์ Content-Type

รูปแบบ Content-Encoding : "Content-Encoding" ":" content-coding

11.4 Content-Length

ใช้บอกขนาดความยาวในบล็อกของข้อมูล มีหน่วยเป็น ไบต์ (Byte) เพื่อที่ผู้รับจะได้ทราบว่ามีข้อมูลส่งมาให้กี่ไบต์ เขียนบอกเป็นตัวเลขฐานสิบ

รูปแบบ Content-Length = "Content-Length" ":" DIGIT

ตัวอย่าง Content-Length : 3495

ความสำคัญของเฮดเดอร์นี้จะใช้ขนาดของบล็อกข้อมูลในการขนถ่ายให้แอปพลิเคชันต่าง ๆ หากไม่ได้ระบุในเฮดเดอร์นี้ เซิร์ฟเวอร์ ปลายทางจะมีรหัสตอบสนองกลับมาเป็น 400(Bad Request)

11.5 Content-Type

ใช้บอกว่าข้อมูลที่อยู่ในบล็อกของข้อมูลเป็นข้อมูลประเภทไหน เช่นหากเป็นเอกสาร HTML จะต้องระบุเป็น text/html ถ้าเป็นไฟล์กราฟิกแบบ gif ต้องระบุเป็น image/gif เป็นต้น

รูปแบบ Content-Type : "Content-Type" ":" media-type

Media-type คือ ลักษณะของข้อมูล ได้แก่ text/html image/gif image/jpg เป็นต้น

รูปแบบของ media-type : type "/" subtype

11.6 Date

บอกวันที่และเวลาที่ขณะเริ่มส่งข้อมูลไปยังผู้รับอาจจะเป็นเซิร์ฟเวอร์หรือไคลเอนต์ก็ได้ กำหนดโดย RFC 822

รูปแบบ Date = "Date" ":" HTTP-Date

ตัวอย่าง Date : Sun, 24 Apr 2000 09:26:22 GMT

11.7 Expires

กำหนดวันที่หมดอายุของไฟล์ที่ส่งไปให้ ไคลเอนต์ รายการนี้ใช้ในทางเทคนิคเพื่อป้องกันการเก็บไฟล์ไว้ในแคช (Cache) จากเว็บเบราเซอร์อย่าง Nagavitor ได้ โดยระบุวันที่ Expires ให้ย้อนจากวันเวลาปัจจุบันนานๆ เมื่อเว็บเบราเซอร์ได้รับไฟล์ไปก็จะเข้าใจว่าไฟล์นั้นหมดอายุแล้ว ถึงแม้จะนำเอาเนื้อหาไฟล์ไปแสดงในเว็บเบราเซอร์แต่ก็ไม่เก็บไว้ในแคช ทำให้ทาง เซิร์ฟเวอร์มั่นใจได้ว่า ทุกครั้งที่ ไคลเอนต์ ร้องขอไฟล์ จะต้องวิ่งมาขอจาก เซิร์ฟเวอร์ ใหม่ทุกครั้ง ถึงแม้จะเป็นการร้องขอไฟล์เดิมๆ ก็ตาม และทาง เซิร์ฟเวอร์ ไม่มีการปรับปรุงไฟล์ก็ตาม

รูปแบบ Expires = "Expires" ":" HTTP-Date

ตัวอย่าง Date.: Sun, 24 Apr 2000 09:26:22 GMT

11.8 From

เฮดเดอร์นี้จะใส่ที่อยู่จดหมายอิเล็กทรอนิกส์ (Internet Mail Address) ของผู้ที่ทำการบริหารหรือควบคุมโปรแกรมที่ทำหน้าที่ร้องขอ เช่น เว็บเบราเซอร์

รูปแบบ From = "From" ":" mailbox

ตัวอย่าง From : webmaster@w3.org

เฮดเดอร์นี้อาจจะใช้ล็อกการร้องขอข้อมูลที่ เซิร์ฟเวอร์ ไม่ต้องการ แต่ไม่ได้หมายความว่าเป็นระบบป้องกันหรือรักษาความปลอดภัยในการเข้าถึงข้อมูล ข้อควรระวังคือไม่ควรให้ไคลเอนต์ เป็นฝ่ายส่งเฮดเดอร์มายัง เซิร์ฟเวอร์เพราะไคลเอนต์สามารถเปลี่ยนแปลงรายละเอียดของเฮดเดอร์ได้

11.9 If-Modified –Since

เฮดเดอร์นี้ใช้กับคำร้องขอเมธอด GET เพื่อสร้างเงื่อนไขบอกแก่ เซิร์ฟเวอร์ว่าถ้าไฟล์ที่ร้องขอไปมีการแก้ไขหลังจากวันที่ได้ระบุในเฮดเดอร์นี้ เซิร์ฟเวอร์ จึงค่อยส่งไฟล์นั้นมาให้ แต่ถ้ายังไม่ได้มีการแก้ไขจากช่วงเวลาที่ได้ระบุ (รหัสตอบสนอง คือ 304- Not modifored) เซิร์ฟเวอร์ไม่ต้องส่งไฟล์นั้นมา เพียงแต่ให้ เซิร์ฟเวอร์ส่งรหัสตอบสนอง 304 มาแทน

รูปแบบ If-Modifired-Since = "If-Modifired-Since" ":" HTTP-Date

ตัวอย่าง Date : Sun, 24 Apr 2000 09:26:22 GMT

11.10 Last-Modifired-Since

เป็นรายการบอกการแก้ไขข้อมูลล่าสุดของไฟล์ที่จะส่งไปให้ ไคลเอนต์

รูปแบบ Last-Modifired-Since = "Last-Modifired-Since" ":" HTTP-Date

ตัวอย่าง Date : Sun, 24 Apr 2000 09:26:22 GMT

10.11 Location

บอกที่อยู่ของแหล่งข้อมูลทรัพยากรที่ไคลเอนต์ ต้องการหากURL ที่ร้องขอมาถูกย้ายไปที่อื่น เซิร์ฟเวอร์ส่งรหัสตอบสนอง 3xx กลับไปยัง ไคลเอนต์ หรือเปลี่ยนทิศทางไปยัง URL ที่ถูกต้อง

รูปแบบ Location = "Location" ":" absoluteURL

ตัวอย่าง Location:http://www.w3.org/hypertext/www/NewLocation.html

11.12 Pragma

ใช้เก็บคำสั่งที่ผู้รับจะนำไปประยุกต์ใช้อีกที คำสั่งต่างๆ จะทำงานภายใต้โปรโตคอล

รูปแบบ Pragma = "Pragma" ":" #pragma-directive

Pragma-directive = "no-cache" | extension-pragma

Etension-pragma = token ["=" word]

เมื่อคำสั่ง no-cache ถูกใช้ในข้อความร้องขอข้อมูลแอปพลิเคชัน จะต้องไปร้องขอมาจาก เซิร์ฟเวอร์ เสมอ ทำให้ ไคลเอนต์ได้รับการตอบสนองต่อ เซิร์ฟเวอร์ จากการร้องขอจริง

11.13 Referer

ให้ไคลเอนต์ระบุ URI ของข้อมูลที่ร้องขอ เฮดเดอร์จะเป็นประโยชน์ต่อ เซิร์ฟเวอร์ที่สามารถลดการใช้แคช ใช้ทรัพยากรของระบบได้น้อยลงไม่ต้องมาค้นหาส่วนการเชื่อมโยง เป็นส่วนที่บอกได้ว่าสามารถเข้าถึงข้อมูลที่ระบุได้หรือไม่

รูปแบบ Referer = "Referer" ":" (absoluteURL | relativeURL)

ตัวอย่าง Referer : http://www.w3.org/hypertext/Database/overview.html

เฮดเดอร์นี้จะไม่ถูกส่งไป หาก ไคลเอนต์ ไม่ได้มาจาก www address, Universal Document Identifirer ,Universal Resource Locators:URL หรือ Universal Resource Names

11.14 Server

เป็นการบอกรายชื่อ ซอฟต์แวร์ ซึ่งทำหน้าที่เป็นเว็บเซอร์เวอร์

รูปแบบ Server = "Server" ":" Program-name/x.xx หรือ

"Server" ":" (Product | comment)

ตัวอย่าง Server Netcape Enterprise/2.01

11.15 User-Agent

เฮดเดอร์ใช้สำหรับบอกรายละเอียดของโปรแกรมที่ทำหน้าที่ส่งข้อความร้องขอในกรณีที่เป็นเว็บเบราเซอร์ ก็ระบุชื่อและเวอร์ชั่นของเว็บเบราเซอร์

รูปแบบ User-Agent = "User-Agent" ":" (Product | comment)

ตัวอย่าง User-Agent : Mozilla/3.0Gold

11.16 www-Authenticate

เฮดเดอร์ใช้ร่วมกับรหัสตอบสนอง 401 (unauthorized) เป็นการบอกว่า เซิร์ฟเวอร์สามารถให้ ไคลเอนต์ เข้าถึงข้อมูลตามที่ ไคลเอนต์ ร้องขอมาได้หรือเปล่า ต้องมีอย่างน้อยหนึ่งตัวคัดค้าน(Challenge) หากมีมากกว่าหนึ่งก็ให้คั่นด้วยเครื่องหมายคอมมา

รูปแบบ www-authorized = "www-authorized""":"challenge

12. Access Authentication

HTTP ได้จัดเตรียมการรับรองการตอบสนองของเซิร์ฟเวอร์ ต่อ ไคลเอนต์ที่ร้องขอมาโดยใช้เครื่องหมายคอมมา คั่นระหว่างค่าสองค่า ร่วมกับรหัสตอบสนอง 401 (unauthorized) การตอบสนองแบบนี้ เซิร์ฟเวอร์ จะส่ง เฮดเดอร์ www-authenticate ไปด้วย

รูปแบบ challenge = auth-scheme 1*SP realm *("," auth-param)

Realm = "realm" "=" realm-value

Realm-value = quoted-string

Auth-scheme = token

Auth-param = token "=" quoted-string

ค่าของ realm จะเป็นการสร้างความปลอดภัยของข้อมูลใน เซิร์ฟเวอร์ ถูกแบ่งส่วนเป็นกลุ่มแล้วกำหนดสิทธิในการขอเข้าใช้งานขอบเขตนี้อาจเป็นชุดของตัวอักษรสร้างขึ้นมาเพื่อกระบวนการนี้โดยเฉพาะ ฝ่ายร้องขอได้รับรหัสตอบสนอง 401 กลับมา ไคลเอนต์ ต้องส่งเฮดเดอร์ไปเพื่อแสดงความน่าเชื่อถือของข้อความร้องขอ ในรูปแบบข้างล่างนี้

Credentials = basic-creentials | (auth-scheme #auth-param)

หากว่าเซิร์ฟเวอร์ไม่ยอมรับจะส่งรหัสตอบสนอง403 (Forbidden) กลับมาขั้นตอนง่ายในการรับรอง คือ ไคลเอนต์ จะส่ง user-id และ password สำหรับ realm เมื่อ เซิร์ฟเวอร์ ยอมรับ จะส่งข้อความตอบสนองกลับไป ตามรูปแบบ ดังนี้

www-authenticate : basic-realm = "WallyWorld"

เมื่อ "WallyWorld" คือชุดของตัวอักษรที่ เซิร์ฟเวอร์ แบ่งจัดกลุ่มไว้สำหรับการร้องขอ รูปแบบของ user-id และ password ของ ไคลเอนต์ มีรูปแบบ ดังนี้

Basic-credentials	= "Basic" SP basic-cookie
Basic-cookie	= <base64 [5] encoding of userid-password, ไม่เกิน 76 ตัวต่อบรรทัด>
Userid-password	= [token] ":" text

เช่น หากทาง โปรแกรมผู้ใช้ ต้องการส่ง user-id = "Aladdin" และ password = "open season" ฝ่ายร้องขอ ต้องส่งเฮดเดอร์ดังนี้

Authorization : Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

ขั้นตอนง่ายๆ ดังกล่าวนี้ ไม่สามารถป้องกันการเขาถึงข้อมูลใน เซิร์ฟเวอร์ จากผู้ใช้ที่ไม่มีสิทธิได้ เพียงแต่สิ่งเหล่านี้จะเป็นพื้นฐานการติดต่อกันระหว่าง ไคลเอนต์ กับ เซิร์ฟเวอร์ ที่จะสร้างสะพานมาเชื่อมต่อกัน

13. Security Consideration

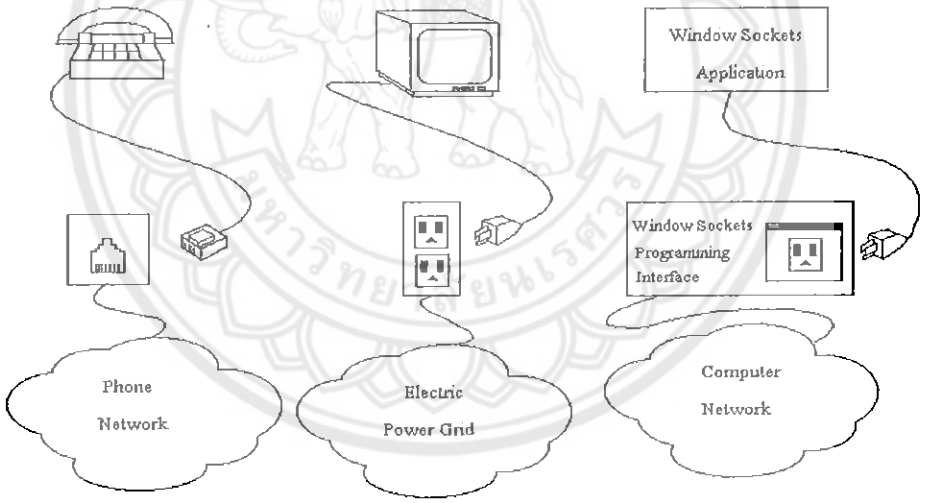
หัวข้อนี้กล่าวถึงระบบการรักษาความปลอดภัย ของ HTTP/1.0 ไม่ได้รวบรวมปัญหาไว้ทั้งหมด แต่จะกล่าวเป็นแนวทางการลดในสิ่งที่ทำให้ความปลอดภัยนั้นลดลง

- ความน่าเชื่อถือจาก ไคลเอนต์ ที่กล่าวมาในหัวข้อ 11 ข้อมูลยังไม่ปลอดภัยป้องกันได้เพียงข้อมูลในบล็อกข้อมูลที่ส่งไปในเน็ตเวอร์กเลเยอร์(Network Layer)ถ้า HTTP/1.0 ไม่ได้มีข้อกำหนดเพิ่มเติมหากว่ามีการเข้ารหัสข้อมูลก่อนส่งก็จะทำให้เพิ่มความปลอดภัยขึ้น
- หลักของความปลอดภัยการใช้โปรแกรมไคลเอนต์ควรจะระมัดระวังว่าจะมีผลต่อเครือข่ายอินเตอร์เน็ตหรือเปล่าควรจะเพิ่มความระมัดระวังให้มากขึ้นกับผลที่อาจจะเกิดโดยไม่ได้คาดไว้ก่อน
- การล็อกข้อมูล เซิร์ฟเวอร์จะต้องอยู่ในจุดที่ข้อมูลมีความปลอดภัยจากการร้องขอของไคลเอนต์ แต่ละประเทศจะมีกฎหมายควบคุมต่างๆ กันไป ผู้คนทั่วไปใช้โปรโตคอล HTTP ในการรวบรวมข้อมูล จะไม่เผยแพร่โดยไม่มีสิทธิ์
- การขนถ่ายข้อมูลเหมือนกับการถ่ายโอนข้อมูลในโปรโตคอลต่างๆ โปรโตคอล HTTP ไม่ได้สร้างกฎระเบียบไว้ ดังนั้นโปรแกรมแอปพลิเคชันควรจะควบคุมข้อมูลของตนเองไว้ด้วยอาจจะใช้เฮดเดอร์ server, referer หรือ From เป็นตัวควบคุมการกระทำภายใต้ไฟล์และทางเดินของไฟล์เซิร์ฟเวอร์ ต้องระวังด้านข้อจำกัดของเอกสาร ที่ส่งให้ไคลเอนต์ ตามข้อความร้องขอเซิร์ฟเวอร์ต้องมีการจัดการมีการป้องกันการเข้าถึงข้อมูลของ ไคลเอนต์ ที่ไม่ให้มีการเข้าถึงได้ เช่น ไฟล์ระบบ ไฟล์ควบคุม หรือ รหัสของข้อมูล เป็นต้น

2.4 วินโดวส์ซ็อกเก็ต (Windows Sockets)

วินซ็อก คือ ระบบเปิดที่เชื่อมต่อการเขียนโปรแกรมระบบเครือข่ายภายใต้ระบบปฏิบัติการวินโดวส์ คำว่าระบบเปิดหมายถึง เป็นระบบที่ผู้พัฒนาโปรแกรมสามารถจะนำวินซ็อกมาใช้ และดัดแปลงโดยที่ไม่ต้องจ่ายเงินค่าลิขสิทธิ์

วินซ็อก หรือ วินโดวส์ซ็อกเก็ตเอพีไอ (Windows Socket API:WSA) รวบรวมฟังก์ชั่น และโครงสร้างข้อมูล ที่จำเป็นต่อการพัฒนาโปรแกรมในระบบเครือข่ายให้ผู้พัฒนาได้นำไปใช้ภายใต้มาตรฐานเดียวกัน ถ้าจะให้เปรียบเทียบแล้ว วินซ็อก เปรียบเสมือน ปลั๊กเสียบสำหรับการเขียนโปรแกรม (programming plug) ที่จะเชื่อมต่อระหว่างโปรแกรม กับระบบเครือข่ายเข้าด้วยกัน เช่นเดียวกันกับ ปลั๊กไฟ ที่เชื่อม ไฟฟ้า กับ เครื่องใช้ไฟฟ้าเข้าด้วยกัน หรือ เปรียบเหมือนกับ แจ็คโทรศัพท์ ที่เชื่อมต่อระหว่างเครื่องโทรศัพท์เข้ากับสายสัญญาณโทรศัพท์



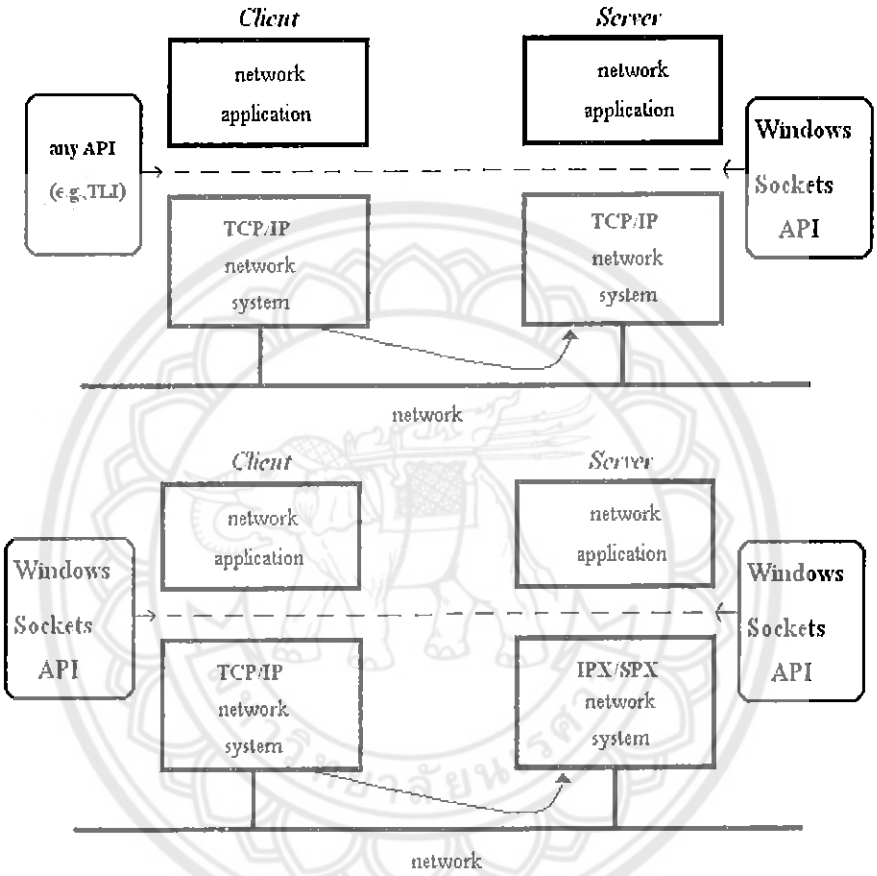
รูปที่ 2.18 เปรียบเทียบหลักการของวินซ็อก

(ที่มา : Windows Socket Network Programming ,Bob Quinn and Dave Shute)

โปรโตคอลและเอพีไอ(Protocols and APIs)

วินซ็อกเอพีไอเป็นอิสระต่อโปรโตคอลวินซ็อกเอพีไอ สามารถติดต่อกับ โปรโตคอลได้หลากหลายชนิด แต่การที่โปรแกรมจะสื่อสารกันได้นั้นจะต้องอยู่บนโปรโตคอลเดียวกัน ดังนั้นหากอยู่บนโปรโตคอลเดียวกันแล้ว ถึงแม้โปรแกรมจะพัฒนามาจากเอพีไอ คนละชุดกันก็สามารถ

ที่จะสื่อสารกันได้ถึงแม้จะไม่ 100 เปอร์เซ็นต์ก็ตาม แต่ถ้าโปรแกรมนั้นอยู่คนละโปรโตคอลกันก็จะไม่มีทางสื่อสารกันได้เลย ถึงแม้จะพัฒนาโปรแกรมมาจากเอพีไอเดียวกันก็ตาม ดังรูปที่ 2.19 ข้างล่างนี้



รูปที่ 2.19 การติดต่อของวินซ็อกเอพีไอกับโปรโตคอลต่างๆ

(ที่มา : Windows Socket Network Programming ,Bob Quinn and Dave Shute)

ในการเชื่อมต่อระหว่างไคลเอนต์กับเซิร์ฟเวอร์นั้น ไคลเอนต์จะต้องรู้ตำแหน่งและรู้จักซ็อกเก็ตของเซิร์ฟเวอร์ และเซิร์ฟเวอร์จะต้องตั้งชื่อซ็อกเก็ตของตัวเองเพื่อให้ไคลเอนต์สามารถใช้อ้างอิงได้ ชื่อของซ็อกเก็ต จะสอดคล้องกับ ไอพีแอดเดรส (IP Address) และ หมายเลขพอร์ต (Port Number)

เมื่อไคลเอนต์ทำการเชื่อมต่อกับเซิร์ฟเวอร์ได้สำเร็จ ซ็อกเก็ตของทั้งสองจะถูกรวมกันอยู่ในรูปแบบของการเชื่อมต่อ ซึ่งประกอบด้วยองค์ประกอบ 5 สิ่งด้วยกันคือ

1. โปรโตคอล (ต้องเป็นโปรโตคอลเดียวกันทั้งไคลเอนต์และเซิร์ฟเวอร์)
2. ไอพีแอดเดรสของไคลเอนต์
3. หมายเลขพอร์ตของไคลเอนต์
4. ไอพีแอดเดรสของเซิร์ฟเวอร์
5. หมายเลขพอร์ตของเซิร์ฟเวอร์

ขั้นตอนการทำงานของโปรแกรมในระบบเครือข่าย

โปรแกรมในระบบเครือข่ายทั้งหมดไม่ว่าจะเป็นไคลเอนต์หรือเซิร์ฟเวอร์ จะมีขั้นตอนการทำงานในการสื่อสารข้อมูล 5 ขั้นตอนดังนี้

เปิดซ็อกเก็ต(Open a socket)

ทั้งไคลเอนต์และเซิร์ฟเวอร์ต้องการซ็อกเก็ตในการสื่อสารข้อมูลในระบบเครือข่าย การเปิดซ็อกเก็ตทำได้โดยใช้ฟังก์ชั่น socket() โดยมีรูปแบบการใช้ฟังก์ชั่นดังนี้

```
SOCKET socket(int af,   /* protocol suite */
              int type,         /* protocol type */
              int protocol);    /*  protocol name */
```

af : “address family”

type : ชนิดของซ็อกเก็ต

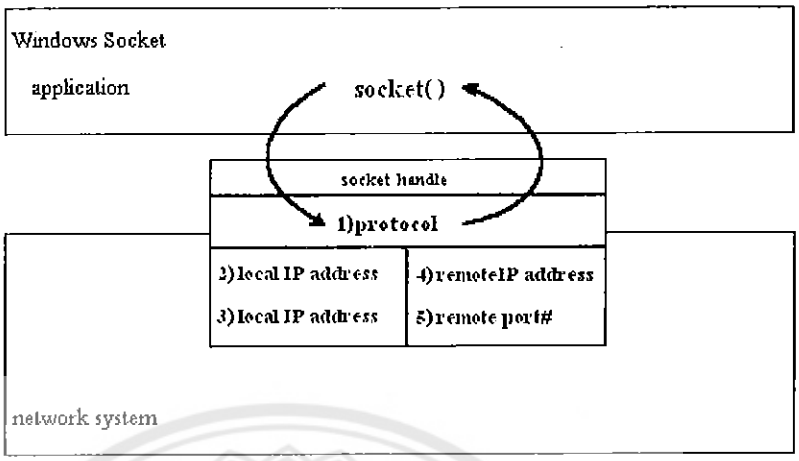
protocol : ชื่อโปรโตคอล

ตัวอย่างเช่น SOCKET socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);

AF_INET เป็นแอดเดรสแฟมิลี่

SOCK_STREAM เป็นซ็อกเก็ตชนิด TCP

IPPROTO_TCP เป็นโปรโตคอลชนิดทีซีพี/ไอพี



รูปที่ 2.20 ขั้นตอนการทำงานของโปรแกรมในระบบเครือข่าย

(ที่มา : Windows Socket Network Programming ,Bob Quinn and Dave Shute)

ฟังก์ชั่น socket() จะคืนค่า socket descriptor เมื่อทำงานสำเร็จ และจะคืนค่าเป็น INVALID_SOCKET เมื่อทำงานผิดพลาด

ให้ชื่อซ็อกเก็ต(Name the socket)

ไคลเอนต์ต้องสามารถรู้ตำแหน่งและรู้จักซ็อกเก็ตของเซิร์ฟเวอร์ และ เซิร์ฟเวอร์ต้องให้ชื่อกับซ็อกเก็ตเพื่อให้ไคลเอนต์สามารถใช้อ้างอิงได้ ซึ่งชื่อของซ็อกเก็ตจะประกอบด้วย 3 สิ่งคือ โปรโตคอล หมายเลขพอร์ต และ แอดเดรส

ในการให้ชื่อซ็อกเก็ต เซิร์ฟเวอร์จะทำการประกาศโครงสร้างข้อมูลซ็อกเก็ตแอดเดรส (Socket address structure) และเรียกฟังก์ชั่น bind()โดยโครงสร้างซ็อกเก็ตและฟังก์ชั่น bind() จะทำการให้ค่าที่อยู่และคุณสมบัติต่าง ๆ กับซ็อกเก็ต

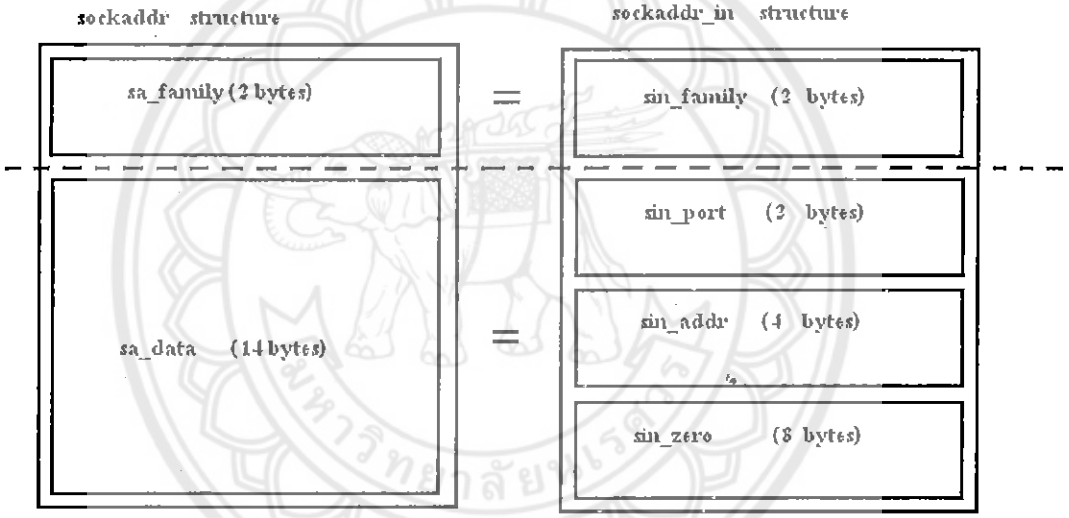
โครงสร้างข้อมูลของซ็อกเก็ต

sockaddr Structure

```
struct sockaddr{
        u_short         sa_family;      /* address family */
        char            sa_data;        /* undefine */
};
```

sockaddr structure เป็นรูปทั่วไปของโครงสร้างข้อมูลซ็อกเก็ตซึ่งในการใช้งานจริงจะไม่อยู่ในรูปแบบนี้ สำหรับการใช้งานกับโปรโตคอลทีซีพี/ไอพีเราจะใช้โครงสร้างข้อมูล sockaddr_in โดยรายละเอียดของโครงสร้างนี้คือ

```
struct sockaddr_in{
    short       sin_family;         /* address family */
    u_short     sin_port;           /*port  number*/
    struct      in_addr  sin_addr;  /*IP address (32 bit) */
    char        sin_zero[8];        /*<unused filler>*/
```



รูปที่ 2.21 หลักการให้ชื่อซ็อกเก็ต

(ที่มา : Windows Socket Network Programming ,Bob Quinn and Dave Shute)

ฟังก์ชั่น bind()

```
int    bind(SOCKET s,           /* an unbound socket */
            struct sockaddr addr,   /* local port and IP address*/
            int  namelen);          /* addr structure length */
```

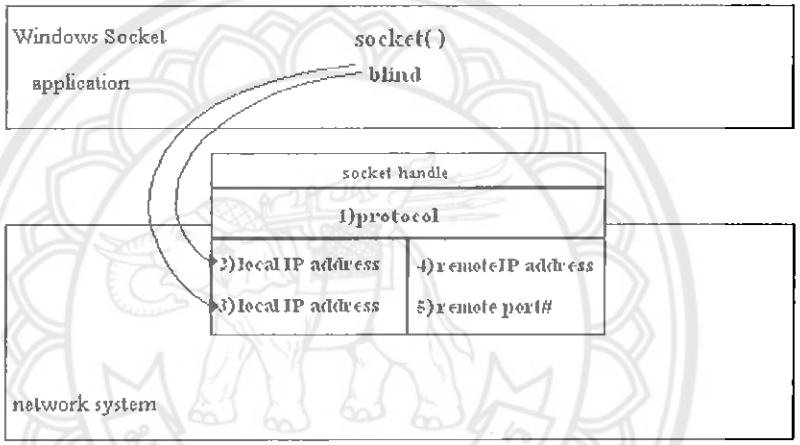
s:socket handle

addr: พ้อยเตอร์(pointer) ที่ชี้ โครงสร้างข้อมูลซ็อกเก็ตแอดเดรส

namelen: ความยาวของโครงสร้างข้อมูลซ็อกเก็ตแอดเดรสที่พ้อยเตอร์ addr เป็นตัวชี้อยู่

ตัวอย่าง

```
SOCKADDR_IN saServer;
SaServer.sin_family = AF_INET;
SaServer.sin_addr.s_addr=INADDR_ANY; //Let WinSock supply address
SaServer.sin_port=htons(nPort); //Use port from command line
int nRet;
nRet=bind(listenSocket,(LPSOCKADDR)&saServer,sizeof(struct sockaddr));
```



รูปที่ 2.22 หลักการใช้ฟังก์ชั่น bind()

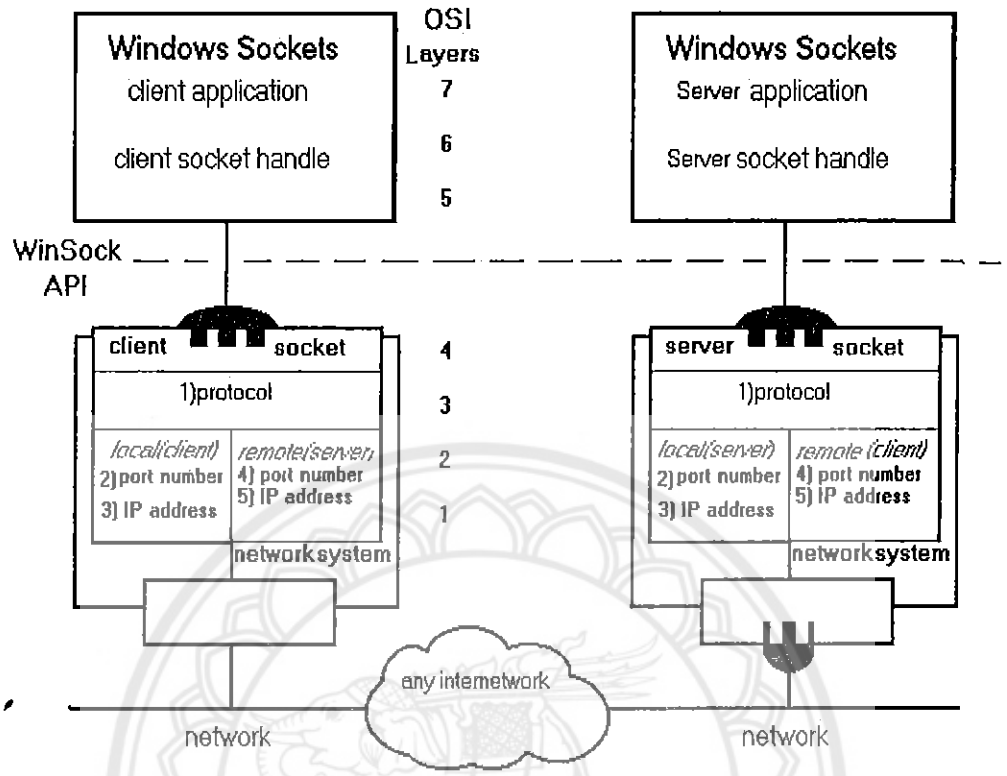
(ที่มา : Windows Socket Network Programming ,Bob Quinn and Dave Shute)

ฟังก์ชั่น bind() จะคืนค่าศูนย์เมื่อทำงานสำเร็จและจะคืนค่า SOCKET_ERROR เมื่อทำงานผิดพลาด สำหรับฟังก์ชั่น bind() นี้จำเป็นต้องเรียกใช้สำหรับโปรแกรมที่เป็นเซิร์ฟเวอร์ แต่สำหรับโปรแกรมที่เป็นไคลเอนต์จะเรียกใช้หรือไม่ก็ได้(ไม่จำเป็น)

สื่อสารกับซ็อกเก็ตอื่น (Associate with another socket)

การสื่อสารระหว่าง 2 ซ็อกเก็ต มีขั้นตอนดังนี้คือ

- เซิร์ฟเวอร์เตรียมการสำหรับการสื่อสาร
- ไคลเอนต์เริ่มการสื่อสาร
- เซิร์ฟเวอร์ตอบสนองการสื่อสาร



รูปที่ 2.23 หลักการของวินซ็อก

(ที่มา : Windows Socket Network Programming ,Bob Quinn and Dave Shute)

หลังจากการสื่อสารสำเร็จ ทั้งไคลเอนต์และเซิร์ฟเวอร์จะรู้จักซ็อกเก็ตของอีกฝ่าย

การเตรียมการสื่อสารของเซิร์ฟเวอร์

สำหรับสตรีมเซิร์ฟเวอร์ (stream server) ซึ่งทำงานในแบบ ทีซีพี คือ ต้องมีการเชื่อมต่อก่อนจึงจะส่งข้อมูลได้ ทางฝั่งเซิร์ฟเวอร์จะรอรับการเชื่อมต่อด้วยฟังก์ชั่น listen()

```
int     listen(SOCKET s,                /* a named,unconnected socket */
               int backlog);            /* pending connect queue length */
```

ตัวอย่าง

```
nRet=listen(listenSocket, //Bound socket
            SOMAXCONN); //Number of connection request queue
```

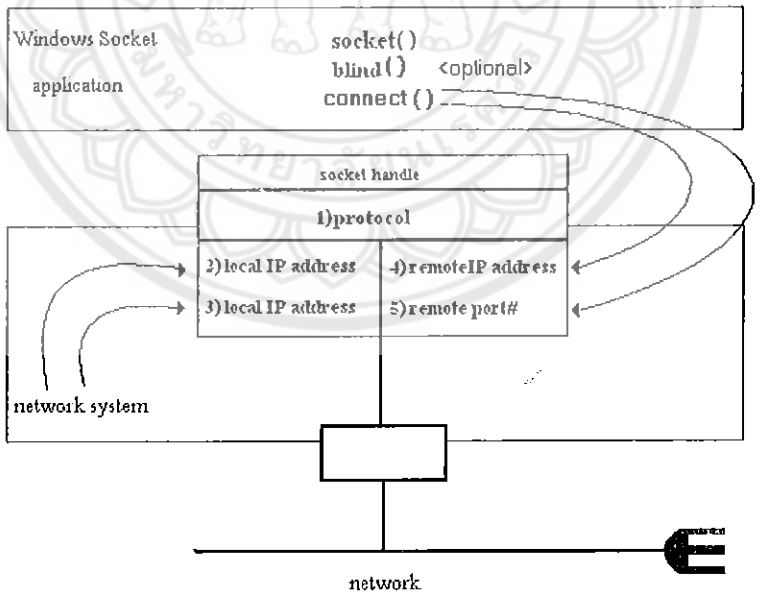
ฟังก์ชั่น listen() จะคืนค่าศูนย์เมื่อทำงานสำเร็จและจะคืนค่า SOCKET_ERROR หากทำงานผิดพลาด

การเริ่มการสื่อสารของไคลเอนต์ สำหรับการเชื่อมต่อแบบทีซีพี ทางฝั่งไคลเอนต์จะต้องเรียกฟังก์ชั่น connect() เพื่อที่จะเริ่มการเชื่อมต่อกับเซิร์ฟเวอร์

```
int connect(SOCKET s,           /* an unconnected socket */
        struct sockaddr *addr,  /*remote port and IP address*/
        int namelen);           /* address structure length */
```

ตัวอย่าง nRet=connect(theSocket,(LPSOCKADDR)&saServer,sizeof(struct sockaddr));

ก่อนที่จะเรียกฟังก์ชั่น connect() เราจะต้องกำหนดค่าให้กับโครงสร้างซ็อกเก็ตแอดเดรสซะก่อน โดยจะต้องกำหนดค่า sin_port และ sin_addr ของ ซ็อกเก็ตทางฝั่งเซิร์ฟเวอร์ (ซ็อกเก็ตของฝั่งตรงข้ามเราจะเรียกว่า remote socket) ฟังก์ชั่น connect() จะทำการระบุที่อยู่ (address) และพอร์ตของซ็อกเก็ตทางฝั่งไคลเอนต์ให้เองดังนั้นฟังก์ชั่น bind() จึงไม่จำเป็นสำหรับโปรแกรมทางฝั่งไคลเอนต์ ฟังก์ชั่น connect() จะคืนค่าศูนย์เมื่อทำงานสำเร็จและจะคืนค่า SOCKET_ERROR เมื่อทำงานผิดพลาด



รูปที่ 2.24 รูปแบบการเชื่อมต่อในระบบเครือข่าย

(ที่มา : Windows Socket Network Programming ,Bob Quinn and Dave Shute)

เซิร์ฟเวอร์ตอบรับการเชื่อมต่อ

สำหรับการเชื่อมต่อแบบทีซีพี เซิร์ฟเวอร์จะทำการตอบสนองต่อฟังก์ชั่น connect() ของไคลเอนต์ ด้วยฟังก์ชั่น accept() ซึ่งจะคอยตรวจสอบการ connect() ของไคลเอนต์

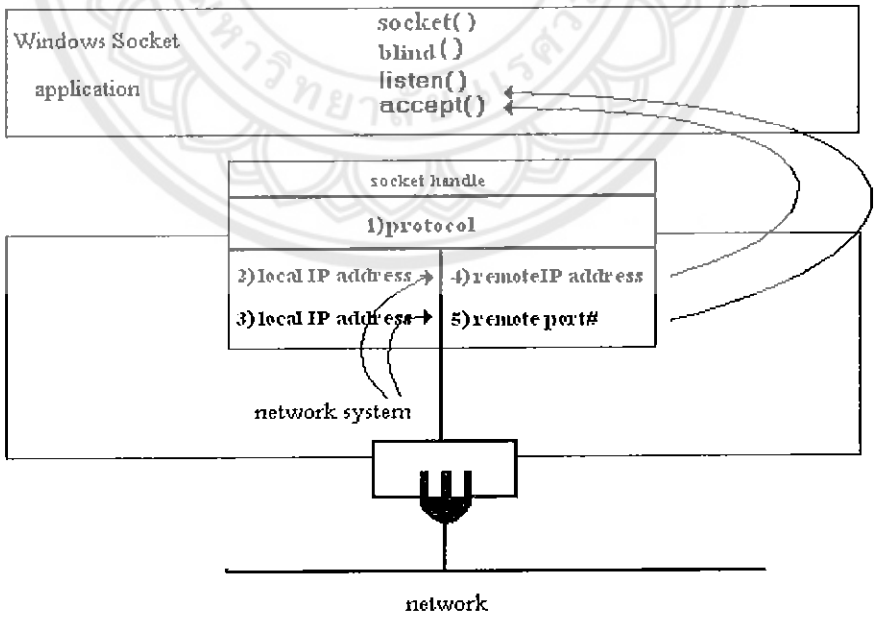
ฟังก์ชั่น accept() คืนค่าเป็นซ็อกเก็ตใหม่หลังจากที่ได้รับการร้องขอการติดต่อจาก ซ็อกเก็ตที่รอการเชื่อมต่อ (listening socket) รูปแบบของฟังก์ชั่น accept() คือ

```
SOCKET accept(SOCKET s,        /* a listening socket */
              struct sockaddr *addr,    /* name of incoming socket */
              int *addrlen);            /*  length of sockaddr */
```

ตัวอย่าง

```
SOCKET remoteSocket;
remoteSocket = accept(listenSocket,      //Listening socket
              NULL,                      //Optional client address
              NULL);
```

ฟังก์ชั่น accept() จะคืนค่าเป็น INVALID_SOCKET ถ้าทำงานผิดพลาด



รูปที่ 2.25 การส่งและรับข้อมูลระหว่างเครือข่าย

(ที่มา : Windows Socket Network Programming ,Bob Quinn and Dave Shute)

ส่งและรับข้อมูลระหว่างซ็อกเก็ต (Send and receive between sockets)

การส่งข้อมูลของซ็อกเก็ตที่ได้ทำการเชื่อมต่อ (connect) เรียบร้อยแล้ว เราจะใช้ฟังก์ชั่น send() นั่นหมายความว่าก่อนที่จะใช้ฟังก์ชั่น send() ได้ จะต้องมีการเรียกฟังก์ชั่นconnect() สำเร็จก่อน รูปแบบของฟังก์ชั่น send() ดังนี้คือ

```
int     send(SOCKET s,                  /* associated socket */
             const char *buf,           /* buffer with outgoing data */
             int len,                   /* bytes to send */
             int flags);                /* option flags */
```

s: socket handle

buf : ตัวชี้ที่ชี้ไปยังข้อมูลที่จะส่งออกไป

len : ความยาวของข้อมูลที่จะส่ง (หน่วยเป็น ไบต์)

flags : สัญญาณที่จะส่ง

การรับข้อมูลเราจะใช้ฟังก์ชั่น recv() ในการรับข้อมูล โดยมีรูปแบบการใช้งานดังนี้

```
int     recv(SOCKET s,          /* associated socket */
             char FAR *buf,     /* buffer with outgoing data */
             int len,           /* bytes to send */
             int flags);        /* option flags */
```

ทั้งฟังก์ชั่น send() และ recv() เมื่อทำงานผิดพลาด จะคืนค่า SOCKET_ERROR และจะคืนค่าเป็นตัวเลขจำนวนไบต์ที่ส่งหรือรับข้อมูล เมื่อทำงานสำเร็จ

ปิดซ็อกเก็ต

ทุกครั้งที่มีการเปิดซ็อกเก็ต เราจะต้องปิดซ็อกเก็ตเมื่อทำงานเสร็จแล้วเสมอ เพื่อไม่ให้เป็นการจองทรัพยากรของเครื่องไปเปล่า ๆ สำหรับการปิดซ็อกเก็ตมีรูปแบบการใช้งานดังนี้

```
int   closesocket(SOCKET s);      /* a  valid socket */
```

Client	Server
Socket()	socket()
Initialize sockaddr_in structure	Initialize sockaddr_in structure
With server(remote) socket name	With server(local) socket name
	bind()
	listen()
Connect() ----------→	
	accept()
< association created , either side can send or receive>	
Send() ----------→	recv()
Recv() ←----------	send()
Closesocket()	Closesocket() (connected socket)
	Closesocket() (listening socket)

รูปที่ 2.26 กลไกการทำงานของไคลเอนต์ - เซิร์ฟเวอร์

(ที่มา : Windows Socket Network Programming ,Bob Quinn and Dave Shute)

จากรูป เป็นรูปแบบการเชื่อมต่อและรับส่งข้อมูล ระหว่างไคลเอนต์ และ เซิร์ฟเวอร์ ที่ทำงานแบบ ทีซีพี

บทที่ 3
ออกแบบและพัฒนาโปรแกรม

เมื่อได้ทำการศึกษาหลักการและทฤษฎีที่เกี่ยวข้องแล้ว ผู้จัดทำก็ได้ทำการกำหนดขอบเขตของการทำโครงงาน คือ โปรแกรมที่พัฒนาขึ้นอ้างอิงตามมาตรฐานอาร์เอฟซี 1945 และสามารถรองรับการทำงานได้ตามมาตรฐานเอชทีทีพี เวอร์ชั่น 1.0

เมื่อกำหนดขอบเขตของการทำโครงงานเรียบร้อยแล้ว ขั้นตอนต่อไปก็คือการออกแบบและพัฒนาโปรแกรม ซึ่งจะอธิบายในหัวข้อถัดไป

3.1 ขั้นตอนการออกแบบโปรแกรม อัลกอริทึมและโครงสร้างโปรแกรม

หลังจากที่ได้ศึกษาและรวบรวมข้อมูล กำหนดขอบเขตของการพัฒนาโปรแกรมเสร็จสิ้นก็นำความรู้ที่ได้มาออกแบบโปรแกรมที่จะทำการพัฒนาขึ้น โดยการทำงานและโครงสร้างของโปรแกรมอธิบายได้ ดังนี้

การทำงานโปรแกรมโดยสังเขป คือ โปรแกรมจะแบ่งการทำงานเป็น 4 ส่วนหลัก ๆ คือ

1. INF หรือ Profile
2. CORE Program
3. ฝ่าย User
4. Storage (Harddisk)

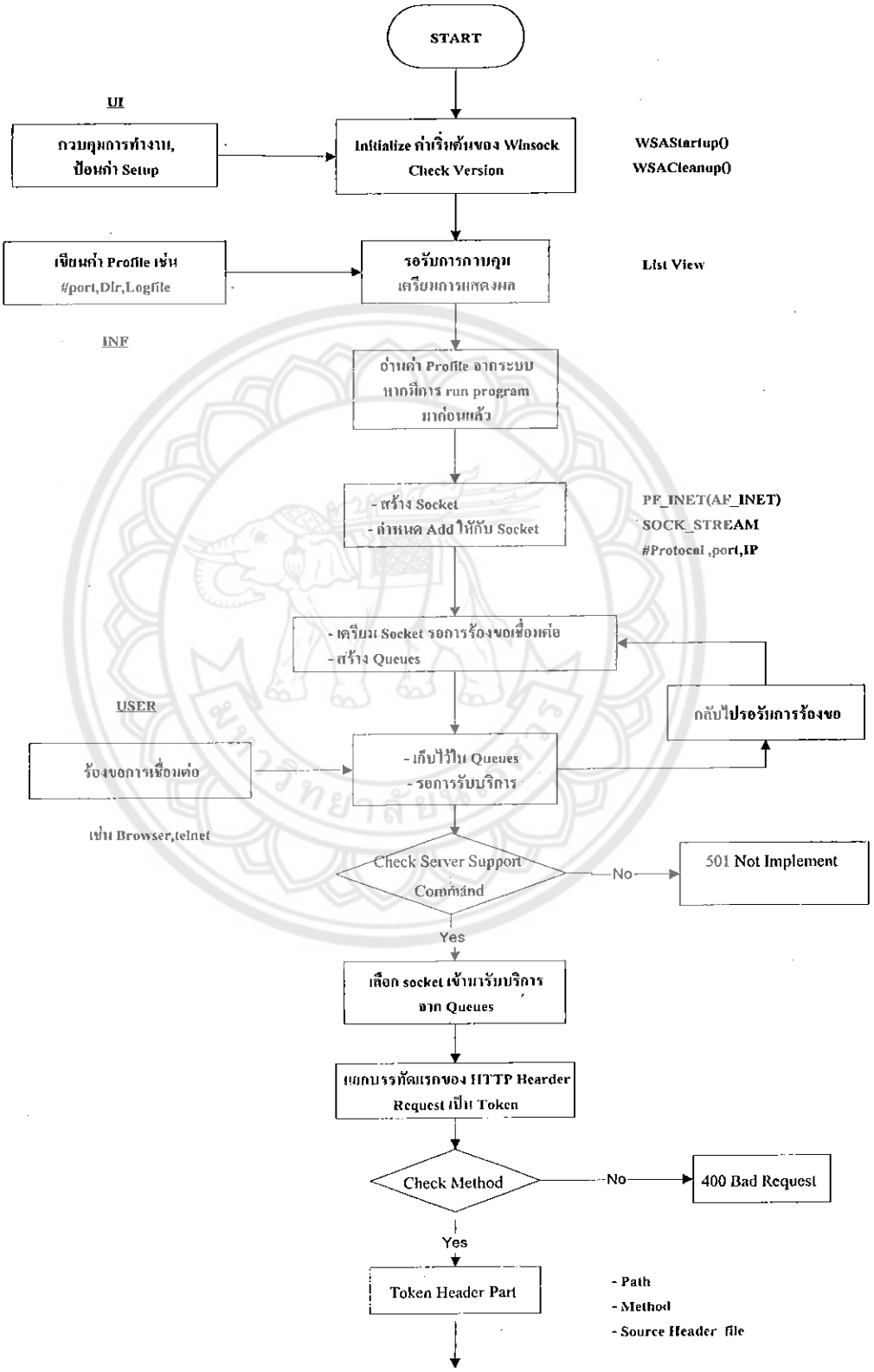
เริ่มต้นการทำงานของโปรแกรม เริ่มจากการ Initialize ค่าเริ่มต้นของวินซ็อก และสร้างวินโดวส์ เพื่อให้เป็น User Interface (UI) ในขณะนี้โปรแกรมจะรอรับคำสั่งจาก Admin ว่าจะให้เริ่มให้บริการหรือยัง เมื่อ Admin สั่งให้โปรแกรมเริ่มทำงาน โปรแกรมก็จะรอให้ Admin กรอกค่า Setup เริ่มต้น เช่น หมายเลขพอร์ต, default document, root folder document เป็นต้น ซึ่งค่าเหล่านี้เมื่อรันโปรแกรมเป็นครั้งที่สอง โปรแกรมจะอ่านค่าProfile มาอัตโนมัติ จากนั้นโปรแกรมจะเตรียมซ็อกเก็ตและรอการร้องขอจากไคลเอนต์(User) ในส่วนนี้โปรแกรมจะแบ่งการทำงาน เป็น 2 ส่วนหลัก คือ ส่วนแรกจะควบคุมการยอมรับการเชื่อมต่อจากไคลเอนต์ แล้วเก็บเข้าไว้ในคิว และอีกส่วนจะเป็นส่วนที่คอยอ่านซ็อกเก็ต จากคิวเข้ามารับบริการ โปรแกรมส่วนที่เป็นฝ่ายเลือกซ็อกเก็ต

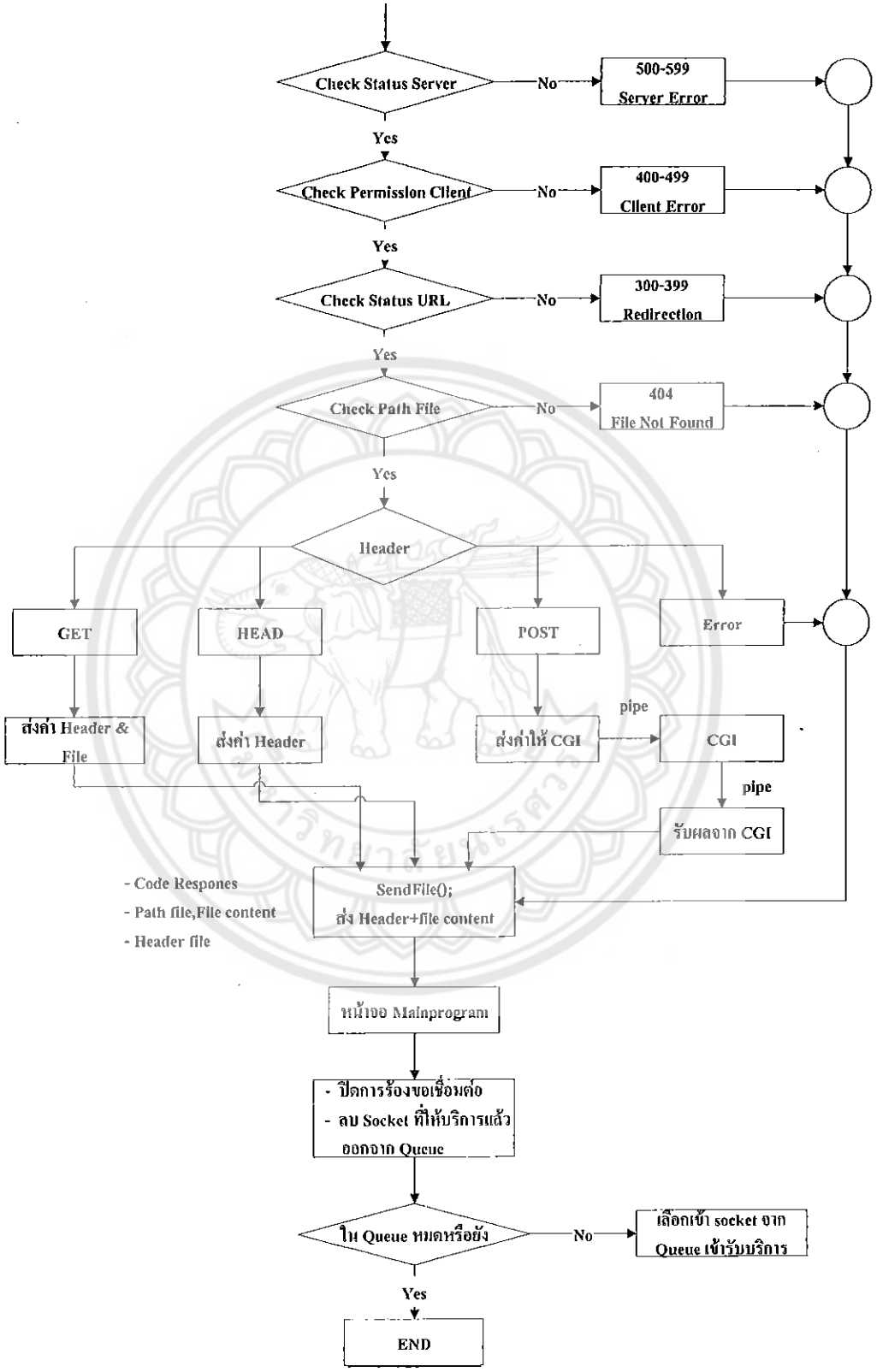
ที่เก็บไว้ในคิวออกมารับบริการ (ซึ่งคิวที่ใช้เก็บซ็อกเก็ต จะเป็นแบบ link list) เมื่อเลือกเข้ามารับบริการแล้ว โปรแกรมจะทำการตัด token บรรทัดแรกของ HTTP Header Request เพื่อนำเอา Type Method และ Path File มาใช้ในการวิเคราะห์ว่า ไคลเอนต์ต้องการทรัพยากรอะไรจากเซิร์ฟเวอร์

ขั้นตอนการวิเคราะห์ มีดังนี้ เริ่มต้นด้วยการตรวจสอบ Method ว่าเป็น GET, HEAD หรือ POST ถ้าไม่ใช่จะถือว่าเป็นคำสั่ง 400 Bad Request ต่อมาจะตรวจสอบสถานะของเซิร์ฟเวอร์, สิทธิของไคลเอนต์, Path File ที่อ้างอิงกับ storage ของเซิร์ฟเวอร์ หากการตรวจสอบนี้ถูกต้องตามเงื่อนไขและกฎเกณฑ์ของ RFC 1945 แล้ว ต่อไปโปรแกรมก็จะรับส่งไฟล์ตามการร้องขอจาก ไคลเอนต์ตามชนิดของ Method ที่ร้องขอ ดังนี้

1. GET โปรแกรมจะสร้าง HTTP Header พร้อมกับทำการส่งเนื้อหาไฟล์ตามที่ ไคลเอนต์ร้องขอกลับไปยังไคลเอนต์
2. HEAD โปรแกรมจะสร้างเพียง HTTP Header กลับไปให้ยังไคลเอนต์
3. POST โปรแกรมจะสร้างไปป์ เชื่อมต่อไปยัง CGI ที่ระบุตาม Path File นั้น แล้วโปรแกรมจะรอรับผลลัพธ์จากการประมวลผลของ CGI ดังกล่าว แล้วทำการสร้าง HTTP Header พร้อมกับส่งผลลัพธ์นี้ไปยัง ไคลเอนต์
4. หากมีคำสั่งนอกเหนือจาก 3 ข้อดังกล่าวนี้จะถือว่าเป็น Error โดยโปรแกรมจะสร้าง HTTP Header และ HTML File ตามชนิด Error ที่เกิดขึ้นไปยังไคลเอนต์ เมื่อโปรแกรมส่งไฟล์ไปยังไคลเอนต์แล้ว โปรแกรมจะปิดการเชื่อมต่อทันที เพื่อไปเลือกเอาซ็อกเก็ต ที่อยู่ในคิวเข้ามาให้บริการต่อไป

การทำงานและโครงสร้างของโปรแกรมแสดงโดยแผนผังลำดับงาน (Flow Chart) ได้ดังนี้





รูปที่ 3.1 แผนผังลำดับงาน (Flow Chart) ของโปรแกรม

3.2 ทำการพัฒนาโปรแกรม

เมื่อได้ออกแบบโปรแกรม อัลกอริทึม และโครงสร้างของโปรแกรมแล้ว ขั้นตอนต่อไปก็คือการพัฒนาโปรแกรม โดยการพัฒนาโปรแกรมนี้ใช้ภาษาวิชวลซีพลัสพลัส เวอร์ชั่น 6.0 และพัฒนาบนระบบปฏิบัติการไมโครซอฟท์วินโดวส์ 98 โดยวิธีการพัฒนาโปรแกรมคือ พัฒนาในแต่ละส่วนย่อยและทดสอบว่ารันผ่านหรือไม่แล้วจึงนำโปรแกรมแต่ละส่วนมารวมกันเป็นโปรแกรมที่สมบูรณ์

เมื่อทำการพัฒนาโปรแกรมเสร็จสิ้นแล้ว ซอร์สโค้ดในส่วนของโปรแกรมหลัก (Main Program) ของโปรแกรมทั้งหมดมีดังนี้

(หมายเหตุ : เนื่องจากซอร์สโค้ดทั้งหมดของโปรแกรมมีความยาวมาก จึงนำเพียงส่วนที่เป็นโปรแกรมหลักมาแสดง ส่วนของซอร์สโค้ดที่สมบูรณ์จะถูกบันทึกอยู่ในซีดีรอม (CD-ROM)

(ซอร์สโค้ดส่วนที่เป็นโปรแกรมหลัก (Main Program))

```
// HTTPUI.C  Hyper Text Transfer Protocol Server
// User Interface Module
// Application/User Interface include

#include "HttpUI.h"
// Message for WinSock async notifications
// and HTTP Server  informational notifications
////////////////////////////////////////////////////////
// Main Program HTTP Server
////////////////////////////////////////////////////////
// Function : WinMain()
// Description: Initial Main dilalog box

int WINAPI WinMain(HINSTANCE hInstance,
                                    HINSTANCE hPrevInstance,
                                    LPSTR lpCmdLine,
                                    int nCmdShow)
```

```
{
        WORD wVersionRequested = MAKEWORD(1,1);
        WSADATA wsaData;
        int nRet;
// Initialize Windows Socket API
        nRet = WSAStartup(wVersionRequested, &wsaData);
        if (nRet)
        {
                MessageBox(NULL,
                                "Initialize Windows Socket Failed",
                                szAppName,
                                MB_OK);
                return 1;
        }

        // Check winsock version
        if (wsaData.wVersion != wVersionRequested)
        {
                MessageBox(NULL,
                                "Wrong Windows Socket Version",
                                szAppName,
                                MB_OK);
                return 2;
        }

        // Use new common controls
           InitCommonControls();
        // Use a dialog as a main control window
           DialogBox(hInstance,
```

```
                    MAKEINTRESOURCE(IDD_MAINWND),
                    NULL,
                    MainControlWndDlg);

    // Release WinSock
    WSACleanup();
    return(0);
}
////////////////////////////////////////////////////
// Function : MainControlWndDlg()
// Description: Control Main dilalog box
// Do all the message processing for the main dialog box
// จัดการเมสเสสโดยรวมของโปรแกรม เช่น ขอร้องขอไฟล์ OK หรือ Not Found หรือ การ
// แสดงผลสถานะของโปรแกรม ที่ Startus Bar

BOOL CALLBACK MainControlWndDlg(HWND hwnd,
                                UINT uMsg,
                                WPARAM wParam,
                                LPARAM lParam)
{
    switch (uMsg)
    {
        HANDLE_DLG_MSG(hwnd, WM_INITDIALOG, OnInitDialog);
        HANDLE_DLG_MSG(hwnd, WM_COMMAND, OnCommand);

        // Pass async messages to HTTP Server
        case UM_ASYNC:
            SocketHandleAsyncMsg(hwnd, wParam, lParam);
            return TRUE;
```

```
                // Handle messages from HTTP Server
                case UM_HTTP:
                        switch(wParam)
                        {
                                case HTTP_FILEOK_MSG:
                                        ShowFileRequested(hwnd, 0, (LPCSTR)lParam);
                                        break;

                                case HTTP_FILENOTFOUND_MSG:
                                        ShowFileRequested(hwnd, 404, (LPCSTR)lParam);
                                        break;

                                case HTTP_EVENT_MSG:
                                        ShowEvent(hwnd, (LPCSTR)lParam);
                                        break;
                        }
                        return TRUE;
        }
        return FALSE;
}
//////////////////////////////////////////////////////
// Function OnInitDialog()
// Initial icon program
// เซ็ตค่าเริ่มต้นของโปรแกรม เช่น Icon , ค่าของ List Views+
BOOL OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)
{
        LV_COLUMN col;
        HWND hwndList;
        TEXTMETRIC tm;
```

```
// Associate an icon with the dialog box.
SetClassLong(hwnd,
                        GCL_HICON,
                        (LONG)LoadIcon(GetWindowInstance(hwnd),
                        MAKEINTRESOURCE(IDI_APP)));

// Setup List Control
// show request resource views
hwndList = GetDlgItem(hwnd, IDC_LIST);
GetTextMetrics(GetDC(hwnd), &tm);
col.mask  = LVCF_FMT|LVCF_TEXT|LVCF_WIDTH;
col.fmt   = LVCFMT_LEFT;
col.pszText = "Hit Time";
col.cx    = tm.tmAveCharWidth*23;
ListView_InsertColumn(hwndList, 0, &col);

col.fmt   = LVCFMT_LEFT;
col.pszText = "In Folder";
col.cx    = tm.tmAveCharWidth*20;
ListView_InsertColumn(hwndList, 1, &col);

col.fmt   = LVCFMT_LEFT;
col.pszText = "Hit Document";
col.cx    = tm.tmAveCharWidth*20;
ListView_InsertColumn(hwndList, 2, &col);

col.fmt   = LVCFMT_LEFT;
col.pszText = "Requestor";
col.cx    = tm.tmAveCharWidth*15;
ListView_InsertColumn(hwndList, 3, &col);
```

```
return(TRUE);
}
////////////////////////////////////////////////////////
// Function : OnCommand()
// Description: Manage message in main control HTTP Service msg
// control start,stop,exit service and about dlg program
// ควบคุมลำดับการทำงานของ UI เช่น จะมีอะไรเกิดขึ้น หากเมื่อมีการ Click ที่ปุ่มต่างที่ Menu Bar

void OnCommand(HWND hwnd, int nId, HWND hwndCtl, UINT codeNotify)
{
        HWND            hwndEvents = GetDlgItem(hwnd, IDC_EVENTLOG);
        static BOOL     fStarted = FALSE;
        HWND            hwndButtonStart;//,hwndButtonStop;
        HTTPSERVINFO    si;
        char            szRootDir[_MAX_PATH];
        int             nRet;
        int             nClear;

        switch (nId)
        {
                case IDC_START:
                        // Start HTTP Server Service
                        hwndButtonStart = GetDlgItem(hwnd, IDC_START);
                        if (!fStarted)
                        {
                                // Get user preferences
                                nRet = DialogBox(GetWindowInstance(hwnd),
                                                MAKEINTRESOURCE(IDD_SERVINFO),
                                                hwnd,
                                                InfoWndProc);
```

```
        if (nRet == IDOK)
    {
        // Fill in the server info
        // get data from profile
        si.nPort = GetProfileInt(szAppName,
                                 szPort,
                                 0);
        GetProfileString(szAppName,
                         szROOTDir,
                         "",
                         szRootDir,
                         sizeof(szRootDir));

        GetProfileString(szAppName,
                         szDefaultPage,
                         "",
                         szDefaultPage,
                         sizeof(szDefaultPage));

/*      GetProfileString(szAppName,
                         szLogFileName,
                         "",
                         szLogFileName,
                         sizeof(szLogFileName));

        // Open logfile, if logging enabled
        //hLogFile = _lcreat (szLogFileName, 0);
        hLogFile = _lopen (szLogFileName, OF_WRITE);
```

```
                    if (hLogFile == HFILE_ERROR) {
                    MessageBox (hwnd,
                            "Unable to open logfile",
                            "File Error",
                            MB_OK | MB_ICONASTERISK);
                    }
*/
                    // Copy szRootDir to show in list views.
                    strcpy(szRootWebDir, szRootDir);
                    // Copy szRootDir to show in list views.
                    strcpy(lpDefaultPage, szDefaultPage);
                    // keep initialize HTTP data

                    si.lpRootDir            = szRootDir;
                    si.hwnd                 = hwnd;
                    si.uMsgAsy              = UM_ASYNC;
                    si.uMsgApp              = UM_HTTP;

                    if (StartServiceHTTP(&si))
                            fStarted = TRUE;

                }
            }
            break;

/*          case IDC_PROPERTIES:
                nRet = DialogBox(GetWindowInstance(hwnd),
                        MAKEINTRESOURCE(IDD_SERVINFO),
                        hwnd,
                        InfoWndProc);
```

```
                    if (nRet == IDOK)
                {
                    // Fill in the server info
                    si.nPort = GetProfileInt(szAppName,
                                             szPort,
                                             0);
                    GetProfileString(AppName,
                                     szROOTDir,
                                     "",
                                     szRootDir,
                                     sizeof(szRootDir));

                    si.lpRootDir            = szRootDir;
                    si.hwnd                 = hwnd;
                    si.uMsgAsy              = UM_ASYNC;
                    si.uMsgApp              = UM_HTTP;

                    fStarted = FALSE;
                }
            break;
*/
        case IDC_STOP:
            // Stop HTTP Server Service
            StopServiceHTTP();
            fStarted = FALSE;

            break;
```

```
case IDC_CLEARVIEWS:
        // Clear views
        for(nClear = 0; nClear < 5 ; nClear++)
                ListBox_DeleteString(hwndEvents, nClear);
        fStarted = TRUE;

        break;

case IDC_HELPCONTENT:
        // Show AboutProgram Dialog
        MessageBox(hwnd,
                "Sorry!,This option does not implement",
                "HTTP Server - CPE2000-05",
                MB_OK+MB_ICONWARNING);
        break;

case IDC_ABOUTDLG:
        // Show AboutProgram Dialog

        DialogBox(GetWindowInstance(hwnd),
                MAKEINTRESOURCE(IDD_ABOUTDLG),
                        hwnd,
                        AboutProgDlg);

        break;
case IDC_EXIT:
        // Exit HTTP Server
        if (!fStarted)
        {
                StopServiceHTTP();
```

```
                    EndDialog(hwnd,0);
                }
                else
                {
                    // If HTTP Server is running, deamon does not
                    // allow you to stop service.
                    MessageBox(NULL,
                        "HTTP Server is on running WWW servicese.\nYou could not exit HTTP Server program until you stop servicese.",
                        "HTTP Server - CPE200-05",
                        MB_OK+MB_ICONWARNING);

                    fStarted = TRUE;
                }
                break;

            case IDCANCEL:
                // If HTTP is running, stop it
                if (fStarted)
                    StopServiceHTTP();

                fStarted = FALSE;
                EndDialog(hwnd, 0);

                break;

    }
}
//////////////////////////////////////////////////////
```

```
// ควบคุมการแสดงผลที่ List Views ว่าแต่ละ Request ที่เข้ามา ทำการร้องขออะไรบ้าง ที่ไหน ชื่อ
อะไร
// เวลาเมื่อไร จากใครเป็นผู้ขอมา
void ShowFileRequested(HWND hwnd, int nError, LPCSTR lpFileName)
{

        char szDisplayName[_MAX_PATH+10];
        HWND hwndList       = GetDlgItem(hwnd, IDC_LIST);
        char                szDateTimeBuff[256];
//      char                hLogFileBuff[1024];
        LV_ITEM                     lvItem;
        int                         nItem;
        SYSTEMTIME          st;

        // Was the file found?
        if (!nError)
                strcpy(szDisplayName, lpFileName);
        else
                wsprintf(szDisplayName,
                                "%d - %s",
                                nError,
                                lpFileName);

        GetLocalTime(&st);
        wsprintf(szDateTimeBuff,"%s, %02d:%02d:%02d.%03d LMT",
                        (LPSTR)__DATE__,
                        st.wHour,   st.wMinute,
                        st.wSecond, st.wMilliseconds);
```

```
// Add a new item to the list
lvItem.mask = LVIF_TEXT;
lvItem.iItem = 0;
lvItem.iSubItem = 0;
lvItem.pszText = szDateTimeBuff;
nItem = ListView_InsertItem(hwndList, &lvItem);

lvItem.iItem = nItem;
lvItem.iSubItem = 1;
lvItem.pszText = szRootWebDir;
ListView_SetItem(hwndList, &lvItem);

lvItem.iItem = nItem;
lvItem.iSubItem = 2;
lvItem.pszText = szDisplayName;
ListView_SetItem(hwndList, &lvItem);

lvItem.iItem = nItem;
lvItem.iSubItem = 3;
lvItem.pszText = szSin_Addr;
ListView_SetItem(hwndList, &lvItem);

/*
if (hLogFile != HFILE_ERROR)
{
        wsprintf(hLogFileBuff,
                "%s %s %s %s %s\n",
                szDateTimeBuff,
                szRootWebDir,
                szDisplayName,
```

```
                    szSin_Addr,
                    szRequestorURL);

    _lwrite (hLogFile, hLogFileBuff, strlen(hLogFileBuff));
    _lclose (hLogFile);

        }
*/

}
////////////////////////////////////////////////////
// Function ShowEvent()
// show events logs of HTTP Server to list-box or tree-control
// แสดงสถานะการทำงานของโปรแกรมที่ Startus Bar เช่น Socket Success,Bind Success,
// Wating for Web Access หรือ Stop Service เป็นต้น

void ShowEvent(HWND hwnd, LPCSTR lpEvent)
{
    char szBuf[284];
    SYSTEMTIME st;
    HWND hwndEvents = GetDlgItem(hwnd, IDC_EVENTLOG);

    GetLocalTime(&st);
    wsprintf(szBuf,"%02d:%02d:%02d.%03d LMT\t%s",
                    st.wHour,   st.wMinute,
                    st.wSecond, st.wMilliseconds,
                    lpEvent);

    SetDlgItemText(hwnd, IDC_LISTEVENT,szBuf);
    ListBox_AddString(hwndEvents, szBuf);
```

```
        if (ListBox_GetCount(hwndEvents) > 500)
                ListBox_DeleteString(hwndEvents, 0);
}
////////////////////////////////////////////////////////
// Function HttpInfoWndProc()
// Get/Write Profile. Make comfortable using.
// Get Profile of initialize setting program to illustrative.
// ควบคุมการทำงานของ Profiles
//   WM_INITDIALOG: โปรแกรมจะทำการอ่านค่าโปรไฟล์มาเก็บไว้ใน Setup Dialog
//ถ้าหากการรันโปรแกรมครั้งนี้ไม่ใช่ครั้งแรก
//      WM_COMMAND:   หากการรันโปรแกรมครั้งนี้  เป็นครั้งแรก  โปรแกรมจะทำการเก็บค่า
port,Directory,
//Default document และ อื่นๆ จาก Setup Dialog เก็บไว้ใน HttpServer.ini

BOOL CALLBACK InfoWndProc(HWND hwnd,
                                        UINT uMsg,
                                        WPARAM wParam,
                                        LPARAM lParam)
{
        char szText[256];
        BOOL fRet = 0;
        switch (uMsg)
        {
                case WM_INITDIALOG:
                        // Fill the controls with previous entries
                        GetProfileString(szAppName,
                                                szPort,
                                                "",
                                                szText,
```

```
                                        sizeof(szText));
    SetDlgItemText(hwnd,
                                    IDC_PORT,
                                    szText);

    GetProfileString(szAppName,
                                        szROOTDir,
                                        "",
                                        szText,
                                        sizeof(szText));
    SetDlgItemText(hwnd,
                                    IDC_ROOTDIR,
                                    szText);

    GetProfileString(szAppName,
                                        szDefaultPage,
                                        "",
                                        szText,
                                        sizeof(szText));
    SetDlgItemText(hwnd,
                                    IDC_DEFAULT,
                                    szText);

    GetProfileString(szAppName,
                                        szLogFileName,
                                        "",
                                        szText,
                                        sizeof(szText));
    SetDlgItemText(hwnd,
                                    IDC_LOGFILE,
```

```
                                        szText);

            fRet = TRUE;

    case WM_COMMAND:
            switch(wParam)
            {
                    case IDOK:
                            // Save the preferences
                            GetDlgItemText(hwnd,
                                                    IDC_PORT,
                                                    szText,
                                                    sizeof(szText));
                            WriteProfileString(szAppName,
                                                    szPort,
                                                    szText);

                            GetDlgItemText(hwnd,
                                                    IDC_ROOTDIR,
                                                    szText,
                                                    sizeof(szText));
                            WriteProfileString(szAppName,
                                                    szROOTDir,
                                                    szText);

                            GetDlgItemText(hwnd,
                                                    IDC_DEFAULT,
                                                    szText,
                                                    sizeof(szText));
                            WriteProfileString(szAppName,
```

```
                                                    szDefaultPage,
                                                    szText);

                                        GetDlgItemText(hwnd,
                                                    IDC_LOGFILE,
                                                    szText,
                                                    sizeof(szText));
                                        WriteProfileString(szAppName,
                                                    szLogFileName,
                                                    szText);

                                        fRet = TRUE;
                                        EndDialog(hwnd, IDOK);
                                        break;

                                    case IDCANCEL:

                                        fRet = TRUE;
                                        EndDialog(hwnd, IDCANCEL);
                                        break;
                                }
                                break;
                        }

                        return fRet;
                }

                //////////////////////////////////////////////////////////
```

```
// Function AboutProgDlg()
// Displays application details,
// Windows socket informations and reference..
// Compiled date
// จัดการข้อมูลในการแสดงผล About Program
BOOL CALLBACK AboutProgDlg(HWND hwnd,
                                        UINT uMsg,
                                        WPARAM wParam,
                                        LPARAM lParam)
{
        char szDataBuff[127];
        switch(uMsg)
        {
        case WM_INITDIALOG:
                wsprintf(szDataBuff,"Compiled: %s, %s\n",
                        (LPSTR)__DATE__,(LPSTR)__TIME__);
                SetDlgItemText(hwnd,IDC_COMPILED,(LPSTR)szDataBuff);
                break;
        case WM_COMMAND:
                switch(wParam)
                {
                case IDOK:
                        EndDialog(hwnd,0);
                        return TRUE;
                }
                break;
        }

        return FALSE;
}
```

บทที่ 4
ผลการทดสอบโปรแกรมและวิเคราะห์ผล

บทที่ 4 นี้ กล่าวถึงการทดสอบโปรแกรมว่ามีขั้นตอนการทดสอบอย่างไรและหลังจากทดสอบโปรแกรมแล้ว ผลการทำงานเป็นอย่างไร รวมถึงการวิเคราะห์ผลการทำงานว่ามีประสิทธิภาพมากน้อยเพียงใด มีข้อบกพร่องหรือต้องปรับปรุงแก้ไขในส่วนใดบ้าง

4.1 จุดประสงค์ของการทดสอบโปรแกรม

1. เพื่อทดสอบโปรแกรมว่าสามารถทำงานได้ผลและมีประสิทธิภาพมากน้อยเพียงใด บรรลุตามวัตถุประสงค์หรือไม่
2. เพื่อทดสอบว่าโปรแกรมสามารถรองรับการให้บริการได้ดีเพียงใด
3. เพื่อหาข้อผิดพลาดของโปรแกรมแล้วทำการปรับปรุงแก้ไขให้ดีขึ้น

4.2 ขั้นตอนการทดสอบการทำงานของโปรแกรม

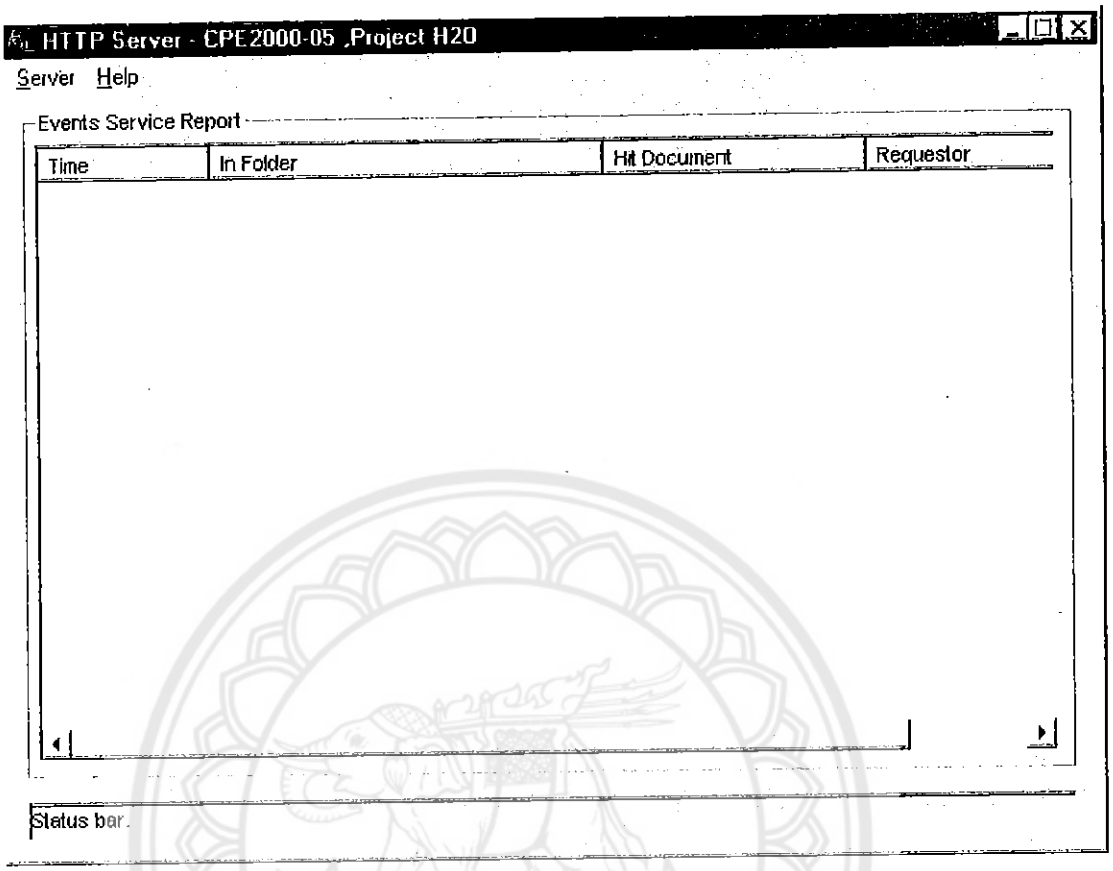
1. ติดตั้งโปรแกรมลงบนเครื่องคอมพิวเตอร์ เพื่อให้เครื่องคอมพิวเตอร์เครื่องนั้น ๆ ทำงานเป็นเซิร์ฟเวอร์ และรอรับการร้องขอจากไคลเอนต์
2. ทดสอบโปรแกรมเพื่อดูผลการทำงานและวัดประสิทธิภาพการทำงาน โดยทำการร้องขอจากไคลเอนต์ไปยังเซิร์ฟเวอร์ที่รันโปรแกรมนี้อยู่ โดยการทดสอบโปรแกรมแบ่งออกเป็น 3 รูปแบบ คือ ทดสอบบนเครื่องคอมพิวเตอร์เดียวกัน ทดสอบโดยใช้คอมพิวเตอร์ 2 เครื่อง และทดสอบบนระบบเครือข่าย (อินทราเน็ต/อินเตอร์เน็ต)
3. ตรวจสอบผลที่ได้จากการทดสอบทั้ง 3 รูปแบบ ว่ามีความถูกต้องและมีประสิทธิภาพการทำงานเป็นอย่างไร ผลการทำงานมีความแตกต่างกันเมื่อทดสอบในรูปแบบที่ต่างกันหรือไม่
4. หาข้อผิดพลาดและทำการปรับปรุงแก้ไข

4.3 ผลการทดสอบโปรแกรม

เมื่อทำการทดสอบโปรแกรมแล้วได้ผลดังนี้

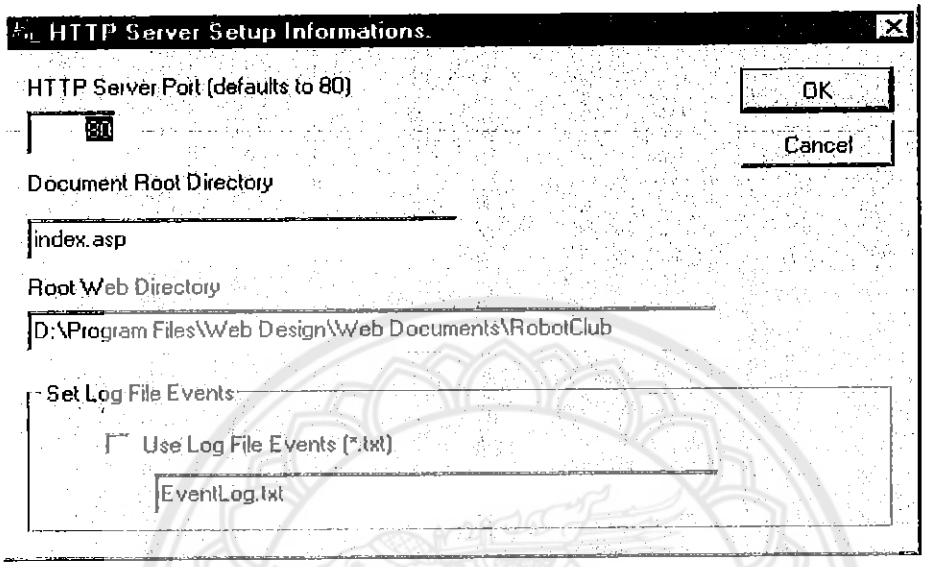
เมื่อเริ่มรันโปรแกรมจะพบหน้าต่าง (Windows) ดังรูปที่ 4.1 เป็นหน้าต่างแรก โดยหน้าต่างนี้จะประกอบด้วย

- เมนู Server
  ประกอบด้วย ส่วนเริ่มต้นการทำงานของการบริการ WWW (Start Service)
  ส่วนหยุดการทำงานของบริการ WWW (Stop Service)
  ส่วนหยุดการทำงานของโปรแกรม (Exit)
- เมนู Help
  ประกอบด้วย ส่วนที่เกี่ยวข้องของโปรแกรม (About HTTP Server 1.0)
- ส่วนการแสดงผลทรัพยากรที่ถูกร้องขอ Events Service Report
  ทำการรายงาน รายการที่ผู้ร้องขอ (client) ขอทรัพยากรของผู้ให้บริการ (server) ได้แก่
  Time : เวลาที่ผู้ให้บริการจัดส่งไฟล์นั้นไปยังผู้ร้องขอ
  In Folder : ห้องแรกของทรัพยากร (root web directory) ที่กำหนดไว้
  Hit Document : รายชื่อทรัพยากรที่ผู้ร้องขอ ขอมา
  Requestor : หมายเลข IP ของผู้ร้องขอทรัพยากร
- ส่วนการแสดงผลสถานะ การทำงานของโปรแกรม (Status Bar)
  แสดงสถานะการทำงานของโปรแกรม เช่น การเชื่อมต่อของ socket, bind, listen, การหยุดการให้บริการ หรือ ค่าผิดพลาดต่างๆ ที่เกิดขึ้นระหว่างการทำงานของโปรแกรม



รูปที่ 4.1 รูปหน้าต่างเพื่อเริ่มใช้งาน

เริ่มการทำงานของโปรแกรมโดยทำการคลิกที่ Server แล้วเลือก start service จะปรากฏหน้าต่างใหม่ขึ้นมาอีกหน้าต่างหนึ่ง เพื่อให้ผู้ใช้กรอกค่าหรือข้อมูลต่าง ๆ ลงไป



รูปที่ 4.2 หน้าต่างเพื่อให้กรอกข้อมูลต่าง ๆ

-TCP Port (defaults to 80)

กำหนดหมายเลขพอร์ตที่ใช้ในการติดต่อสื่อสารผ่านทางโปรโตคอล HTTP ปรกติกำหนดค่าไว้เป็น 80

-Document Root Directory

กำหนดหน้าแรกของเอกสาร หรือที่เรียกว่าโฮมเพจ ที่ต้องการแสดง เช่น index.html, index.asp,index.shtml, index.php4 เป็นต้น

-Root Web Directory

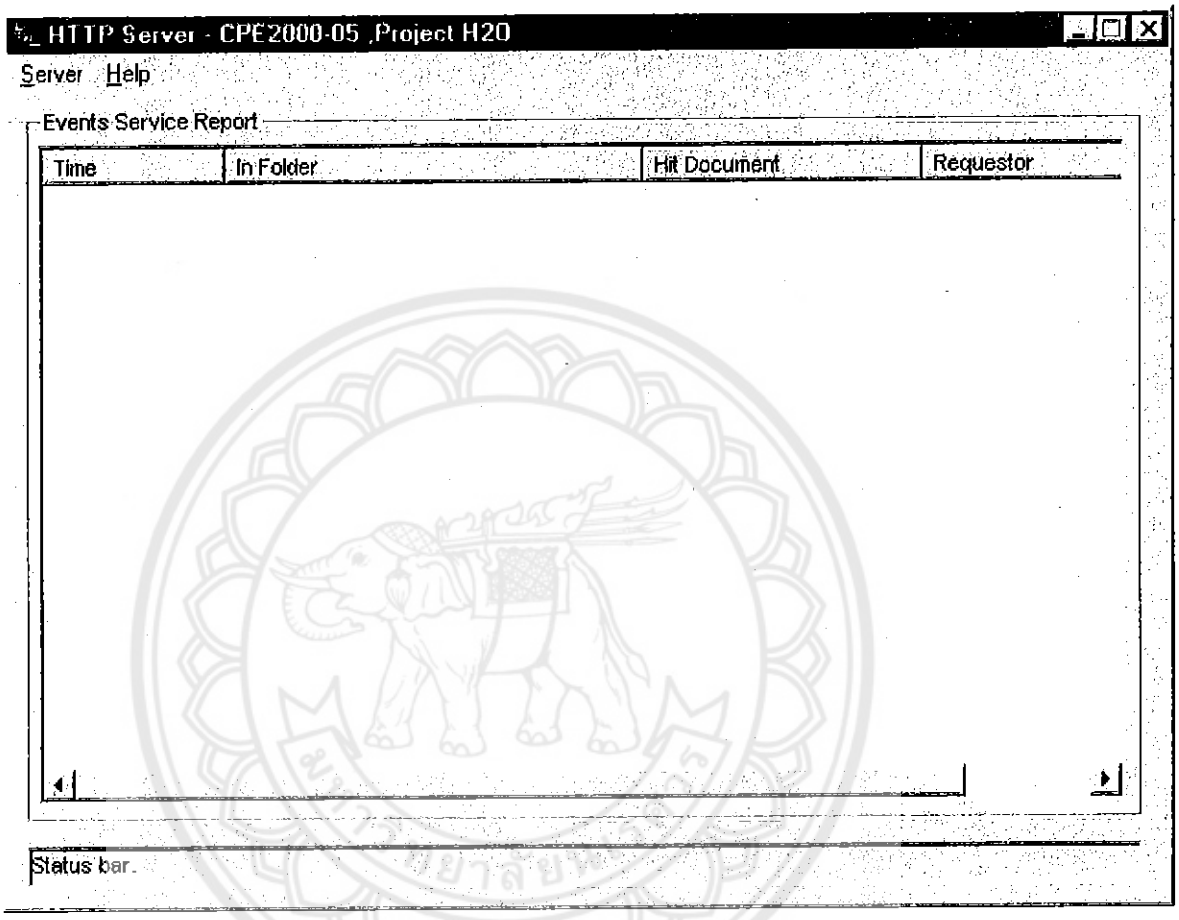
กำหนดรากของเว็บเพ็จ ที่เราต้องการจะให้บริการ

-Set Log File Events

กำหนดให้มีการเขียนเหตุการณ์ ที่เกิดขึ้นขณะให้บริการเอกสาร เช่น

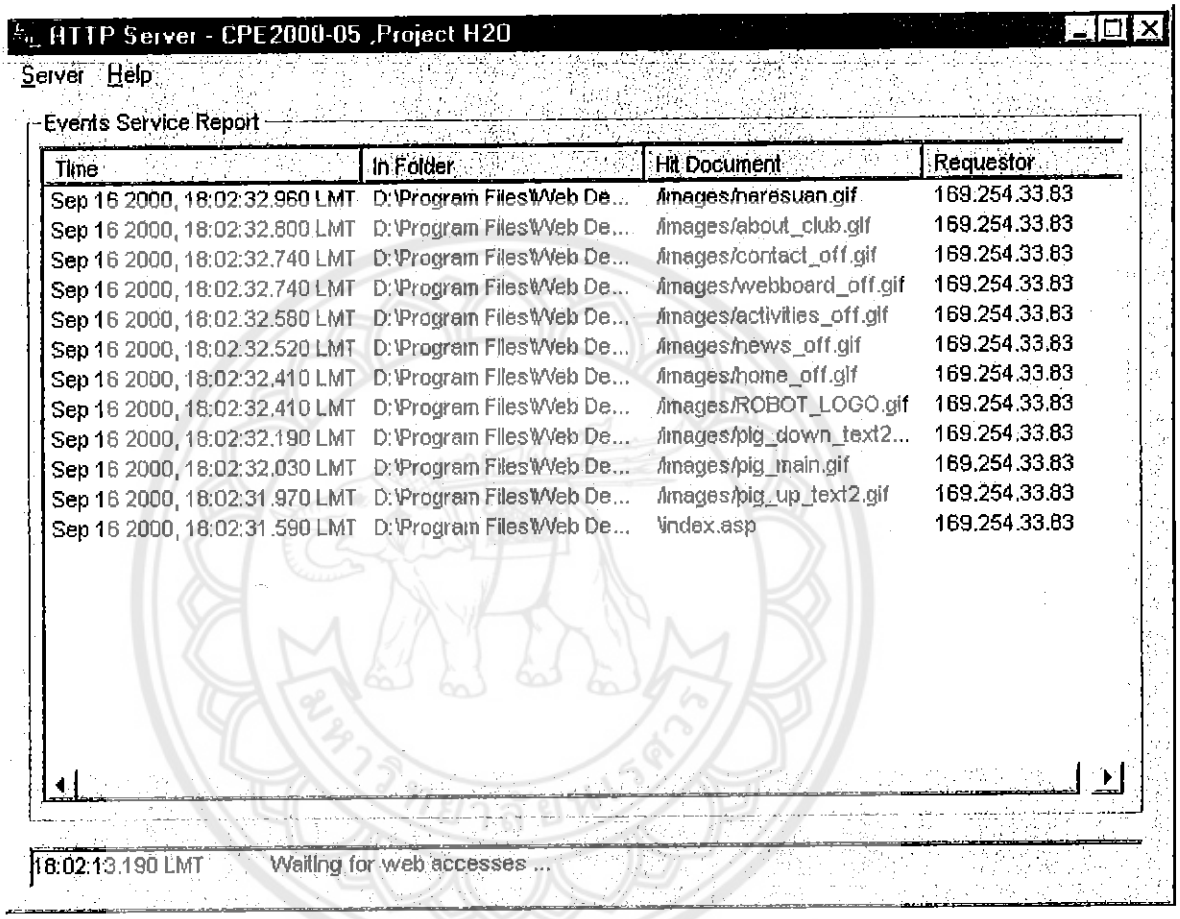
เขียน ชื่อเอกสาร, ที่อยู่ของเอกสาร, ผู้ร้องขอเอกสารนั้น, เวลาที่ทำการร้องขอเอกสาร เป็นต้น

เมื่อกรอกข้อมูลต่าง ๆ ที่ต้องการเรียบร้อยแล้วคลิก OK จะปรากฏหน้าต่างเดิมขึ้นมาอีกครั้ง ดังรูป ในสภาวะนี้โปรแกรมจะเปิดให้บริการเรียบร้อยแล้วและรอการร้องขอจากไคลเอนต์อยู่



รูปที่ 4.3 รูปหน้าต่างหลังของโปรแกรม HTTP Server – CPE2000-05

หากมีการร้องขอจากไคลเอนต์เข้ามาเพื่อขอรับบริการทรัพยากรจากโปรแกรม และเซิร์ฟเวอร์มีข้อมูลนั้นอยู่ โปรแกรมก็จะให้บริการส่งทรัพยากรนั้นกลับไปให้ไคลเอนต์ และจะมีการแสดงผลการทำงานของโปรแกรม (ผลที่ฝั่งเซิร์ฟเวอร์) ดังรูป



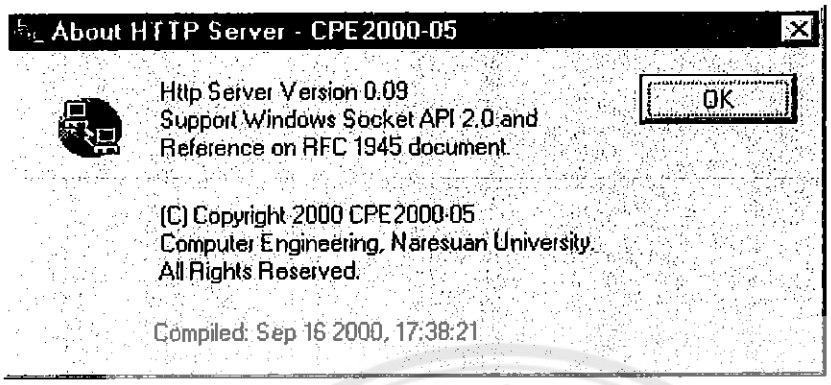
รูปที่ 4.4 ลักษณะ Events Service Report แสดงรายละเอียดเมื่อมีการร้องขอทรัพยากรจากผู้ร้องขอ

ส่วนไคลเอนต์เมื่อได้รับการบริการจากเซิร์ฟเวอร์ คือ ได้รับทรัพยากรตามที่ได้ร้องขอจากเซิร์ฟเวอร์ได้ผลดังรูปที่ 4.5 (ในการทดสอบนี้ไคลเอนต์ คือ เว็บเบราเซอร์ Internet Explorer)

รูปที่ 4.5 ผลการร้องขอที่ฝั่งไคลเอนต์

หากต้องการที่จะปิดการให้บริการของโปรแกรม (เซิร์ฟเวอร์) ก็คลิกที่ Server และเลือกที่ stop service ไคลเอนต์ก็จะไม่สามารถติดต่อและร้องขอทรัพยากรจากเซิร์ฟเวอร์ได้

หากมีการคลิกที่ Help ก็จะปรากฏรายละเอียดของโปรแกรมดังรูป



รูปที่ 4.6 หน้าต่างที่แสดงรายละเอียดของโปรแกรม

4.4 วิเคราะห์ผล

จากผลการทดสอบโปรแกรมปรากฏว่า ผลการทำงานของการทดสอบทั้ง 3 รูปแบบไม่แตกต่างกัน คือให้ผลการทำงานที่เหมือนกันที่ทุกประการ แสดงว่ารูปแบบหรือสภาวะแวดล้อมของระบบเครือข่ายไม่มีผลต่อการทำงานของโปรแกรม นั่นคือ ไม่ว่าเซิร์ฟเวอร์และไคลเอนต์จะอยู่บนเครื่องเดียวกันอยู่คนละเครื่องหรืออยู่บนระบบเครือข่ายอินทราเน็ต/อินเตอร์เน็ตโปรแกรมสามารถให้บริการตามคำร้องขอทรัพยากรจากไคลเอนต์ได้เช่นเดียวกัน และโปรแกรมสามารถให้บริการการร้องขอจากไคลเอนต์หลาย ๆ เครื่องได้พร้อม ๆ กัน และไม่จำเป็นต้องเป็นการร้องขอจากไคลเอนต์ในรูปแบบเดียวกัน เช่น เป็นการร้องขอจากไคลเอนต์ที่อยู่เครื่องอื่น ๆ พร้อม ๆ กับการร้องขอจากไคลเอนต์ที่อยู่บนเครื่องเดียวกันก็ได้

บทที่ 5
สรุปผลและข้อเสนอแนะ

ในบทนี้เป็นการสรุปผลทั้งหมดของการทำโครงงานนี้ ซึ่งประกอบด้วยส่วนสรุปผล ปัญหาในการทำงาน ข้อเสนอแนะ และแนวทางในการพัฒนาต่อไป หากมีผู้ที่สนในที่จะนำโครงงานนี้ไปพัฒนาและปรับปรุงให้มีประสิทธิภาพดีขึ้นต่อไป

5.1 สรุปผล

1.โปรแกรมสามารถให้บริการการร้องขอจากเครื่องลูกข่าย (Client) แบบ GET ได้ คือ การร้องขอไฟล์จากเครื่องแม่ข่าย และแบบ HEAD คือ การร้องขอข้อมูลในส่วนเฮดเดอร์ของไฟล์จากเครื่องแม่ข่ายได้

2. โปรแกรมที่พัฒนายังไม่สามารถให้บริการการร้องขอแบบ POST คือ เป็นการร้องขอเพื่อที่จะส่งข้อมูลจากเครื่องลูกข่ายไปยังเครื่องแม่ข่ายได้ เนื่องจากการร้องขอแบบ POST เป็นการร้องขอที่ต้องทำงานร่วมกับ CGI ทำให้มีความซับซ้อนในการพัฒนามากขึ้น ทำให้ผู้จัดทำโครงงานไม่สามารถพัฒนาโปรแกรมได้ทันกับเวลาที่จำกัด

3. โปรแกรมที่พัฒนาขึ้นสามารถนำไปใช้รันบนเครื่องแม่ข่าย เพื่อให้บริการเว็บ (web) แก่เครื่องลูกข่ายได้ด้วยประสิทธิภาพระดับหนึ่ง โดยอ้างอิงมาตรฐาน เอชทีทีพี เวอร์ชั่น 1.0 (HTTP version 1.0) และอาร์เอฟซี 1945 (RFC 1945)

4.โปรแกรมมีส่วนติดต่อกับผู้ใช้(User Interface) ในลักษณะหน้าต่างทำให้สะดวกและง่ายต่อการใช้งานมากขึ้น

5.2 ปัญหาในการทำงาน

1. การพัฒนาโปรแกรมนี้ใช้ภาษาวิชวลซีพลัสพลัส เวอร์ชั่น 6.0 (Visual C++ version 6.0) ในการพัฒนา ซึ่งเป็นภาษาที่ผู้จัดทำไม่คุ้นเคยในการใช้งานมาก่อน ทำให้ต้องใช้เวลาในการศึกษาการใช้งานภาษาวิชวลซีพลัสพลัส เวอร์ชั่น 6.0 (Visual C++ version 6.0) มาก มีผลทำให้การพัฒนาโปรแกรมเป็นไปอย่างล่าช้า

2. การพัฒนาโปรแกรมในส่วนการร้องขอแบบ POST ไม่สามารถทำได้ทัน เนื่องจากมีความซับซ้อนของระบบการทำงาน คือ ต้องมีการทำงานร่วมกับ CGI โดยมีลำดับการทำงานดังนี้ เมื่อเครื่องแม่ข่ายได้รับการร้องขอแบบ POST แล้ว ต้องส่งต่อไปยัง CGI เพื่อทำการประมวลผลก่อน เมื่อ CGI ประมวลผลเสร็จแล้ว จะส่งผลกลับไปให้เครื่องลูกข่าย ด้วยความซับซ้อนของการทำงาน จึงทำให้การพัฒนาโปรแกรมไม่สามารถทำได้ทันกับเวลาที่กำหนด

5.3 ข้อเสนอแนะ

1. ควรพัฒนาโปรแกรมให้เสร็จสิ้นตามวัตถุประสงค์ เพื่อให้ได้ตามมาตรฐานที่อ้างอิง คือ สามารถให้บริการการร้องขอได้ 3 แบบ คือ แบบ GET , HEAD และ POST

2. ควรดำเนินงานแต่ละขั้นตอนให้ได้ตามระยะเวลาที่กำหนด เพื่อไม่ให้เกิดความล่าช้าในการทำโครงงานตลอดจนต้องศึกษาข้อมูลให้ถ่องแท้เสียก่อนเพื่อไม่ให้เกิดข้อผิดพลาดขึ้นในการทำงาน

3. ควรศึกษาการทำงานของ CGI ตลอดจนข้อมูลอื่น ๆ เพื่อจะนำไปใช้ในการปรับปรุงแก้ไขโปรแกรมในโอกาสต่อไป

5.4 แนวทางในการพัฒนา

1. ศึกษาข้อมูลและการทำงานของ CGI แล้วนำมาพัฒนาโปรแกรมในส่วนที่เหลือ คือ การร้องขอแบบ POST

2. เพิ่มประสิทธิภาพการทำงานในส่วนต่าง ๆ ของโปรแกรม เช่น ความปลอดภัยของข้อมูลที่ได้รับการร้องขอว่าจะอนุญาตให้ใช้ได้หรือไม่ ความสามารถในการให้บริการ เป็นต้น

3. พัฒนาโปรแกรมเพิ่มเติมเพื่อให้โปรแกรมสามารถรองรับการทำงานของ HTTP version ที่สูงขึ้น เช่น เวอร์ชั่น 1.1 ซึ่งจะมีวิธีการร้องขอทรัพยากร (method) ที่มากขึ้น เช่น เมธอด PUT

บรรณานุกรม

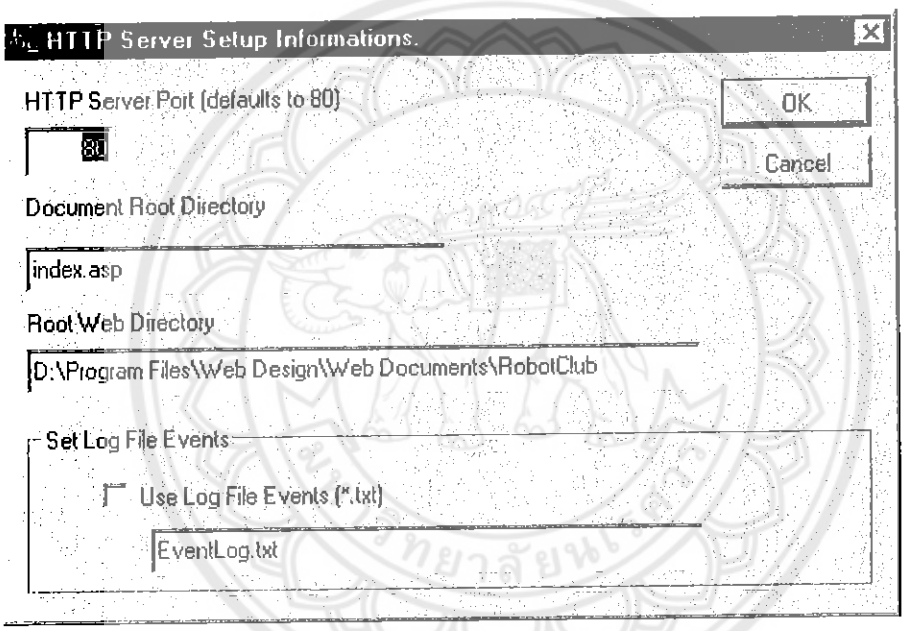
1. ฉลองชัย จงประเสริฐพร. CGI WEB PROGRAMMING.กรุงเทพฯ:วิตตี้ กรุ๊ป จำกัด.
2. ทรงเกียรติ ภาวดี. แกะรอย CGI.กรุงเทพฯ:วิตตี้ กรุ๊ป จำกัด,2542
3. นิรุธ อำนวยศิลป์. คู่มือการเขียน Microsoft Visual C++ Version 6.0 ฉบับเพื่อใช้งานจริง. กรุงเทพฯ:ซัคเซส มีเดีย จำกัด.พิมพ์ครั้งที่ 3.
4. พิเชษฐ์ วงศ์เมธานุเคราะห์. การพัฒนาโปรแกรมประสานระหว่างโปรแกรมประยุกต์ผ่านระบบวินโดว์.กรุงเทพฯ:บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย,2539
5. วิกกิ้น,ดับบลิว ริชาร์ด. THE INTERNET FOR EVERYONE.กรุงเทพฯ: แมคกรอฮิลล์,2539
6. สัจจะ จรัสรุ่งรวีจร. Internet Programming.นนทบุรี:อินโฟเพรส,2542
7. สุวัฒน์ ปุณณชัย และคณะ. เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต.กรุงเทพ: โปรวิชั่น,2543
8. Bob Quinn and Dave Shute. WINDOWS SOCKET NETWORK PROGRAMMING.Massachusetts:Addison-Wesley,First Printing,1995

ภาคผนวก

ภาคผนวก ก

วิธีการใช้การงานโปรแกรม HTTP Server - CPE2000-05.

1. คลิกที่เมนู Server
2. คลิกที่ Start Service จะปรากฏ Dialog Box ของ HTTP Server Setup Informations. ขึ้นมา



3. ทำการกรอกรายละเอียด ดังนี้

-TCP Port (defaults to 80)

กำหนดหมายเลขพอร์ตที่ใช้ในการติดต่อสื่อสารผ่านทางโปรโตคอล HTTP ปรกติกำหนดค่าไว้เป็น 80

-Document Root Directory

กำหนดหน้าแรกของเอกสาร หรือที่เรียกว่าโฮมเพจ ที่ต้องการแสดง เช่น index.html, index.asp,index.shtml, index.php4 เป็นต้น

-Root Web Directory

กำหนดรากของ เว็บเพ็จ ที่เราต้องการจะให้บริการ

-Set Log File Events

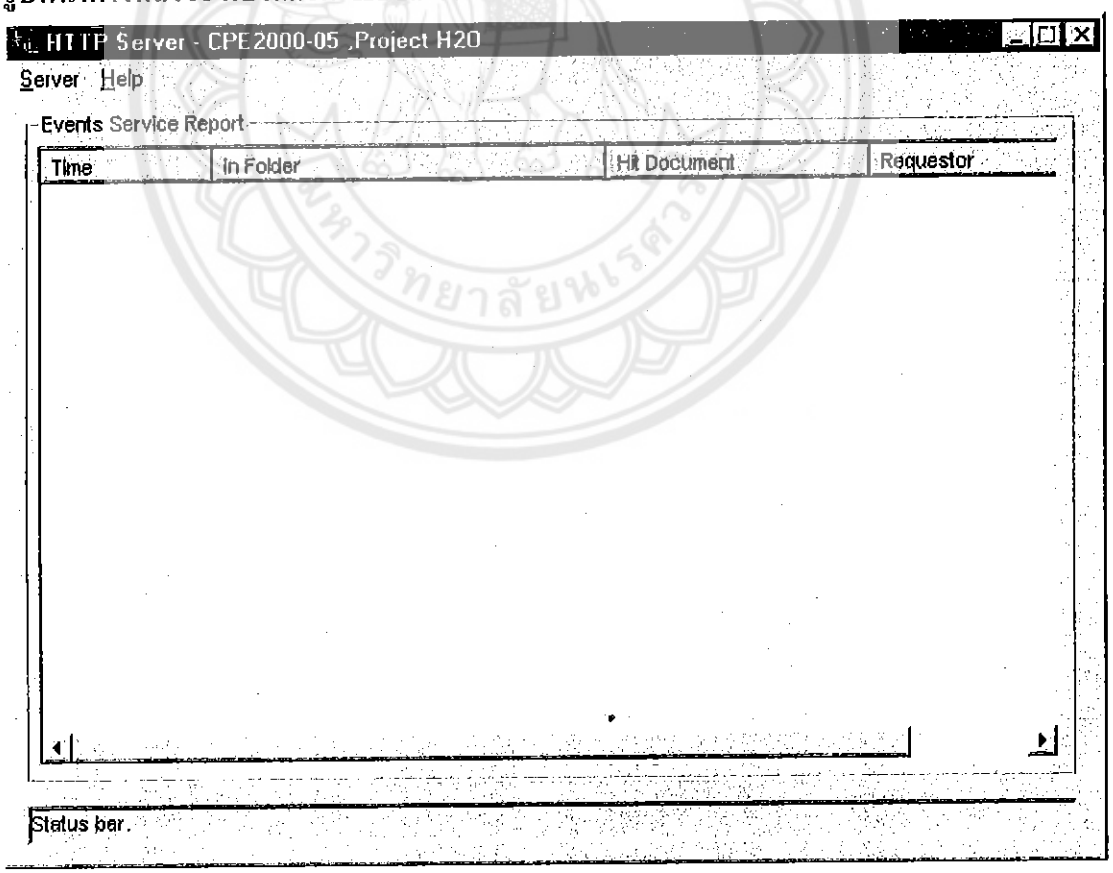
กำหนดให้มีการเขียนเหตุการณ์ ที่เกิดขึ้นขณะให้บริการเอกสาร เช่น เขียน ชื่อเอกสาร, ที่อยู่ของเอกสาร, ผู้ร้องขอเอกสารนั้น, เวลาที่ทำการร้องขอเอกสาร เป็นต้น

* หัวข้อ Set Log File Events ยังไม่สามารถพัฒนาได้ทัน จึงเก็บได้เฉพาะเหตุการณ์ปัจจุบัน

** ค่าที่ทำการกรอกในข้อ 3. โปรแกรมจะทำการเขียน Profile เก็บไว้ เพื่อความสะดวกในการ เซ็ตค่าในข้อ 3.ในโอกาสต่อไปเมื่อกรอกข้อมูลเสร็จแล้ว คลิกปุ่ม OK

4. เมื่อคลิกปุ่ม OK แล้ว โปรแกรมก็จะทำการเตรียมความพร้อมเพื่อรอการเชื่อมต่อจากผู้ร้องขอข้อมูล สังเกตได้ที่ Status Bar ว่า "Waiting for web accesses ..." ตอนนี้ โปรแกรมพร้อมที่จะให้บริการ รอการร้องขอจากผู้ร้องขอ
5. เมื่อต้องการจะเลิกให้บริการ คลิกที่ Stop Service โปรแกรมก็จะหยุดให้บริการ
6. หากเมื่อต้องการให้โปรแกรม สามารถให้บริการได้อีก ให้ทำตามข้อ 3.
7. หากเมื่อต้องการเลิกการใช้โปรแกรม คลิกที่ Exit

รูปหน้าต่างหลังของโปรแกรม HTTP Server – CPE2000-05



ประกอบด้วย

- เมนู Server

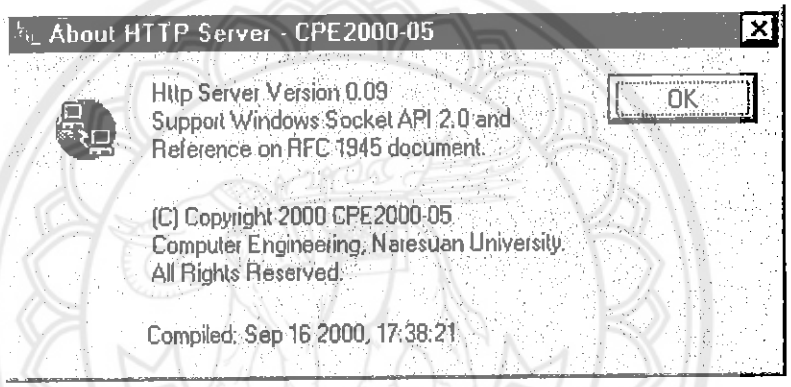
  ประกอบด้วย ส่วนเริ่มต้นการทำงานของการบริการ WWW (Start Service)

  ส่วนหยุดการทำงานของบริการ WWW (Stop Service)

  ส่วนหยุดการทำงานของโปรแกรม (Exit)

- เมนู Help

  ประกอบด้วย ส่วนที่เกี่ยวข้องของโปรแกรม (About HTTP Server 1.0)



- ส่วนการแสดงผลทรัพยากรที่ถูกร้องขอ Events Service Report

  ทำการรายงาน รายการที่ผู้ร้องขอ (client) ขอทรัพยากรของผู้ให้บริการ (server) ได้แก่

  Time : เวลาที่ผู้ให้บริการจัดส่งไฟล์นั้นไปยังผู้ร้องขอ

  In Folder : ห้องแรกของทรัพยากร(root web directory) ที่กำหนดไว้

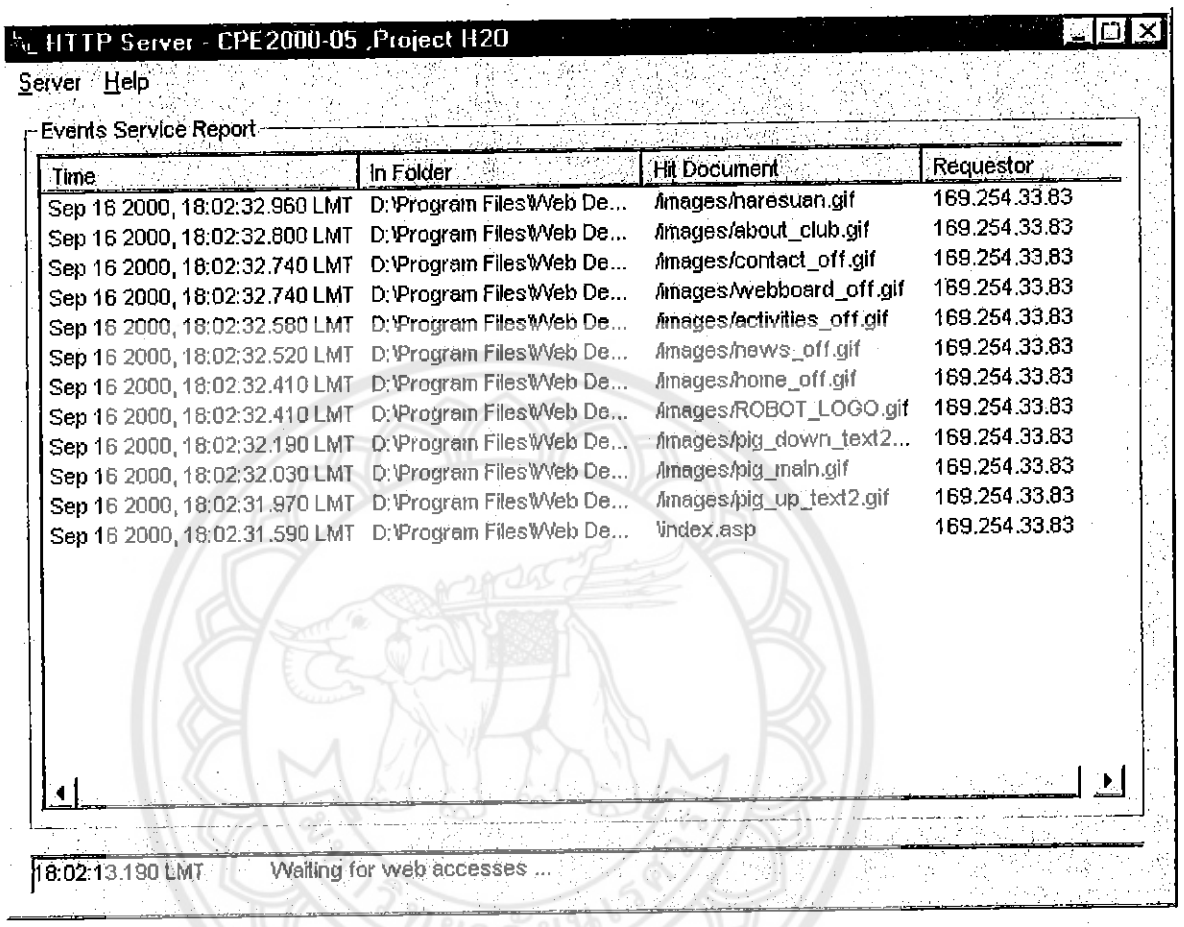
  Hit Document : รายชื่อทรัพยากรที่ผู้ร้องขอ ขอมา

  Requestor : หมายเลข IP ของผู้ร้องขอทรัพยากร

- ส่วนการแสดงผลสถานะ การทำงานของโปรแกรม (Status Bar)

  แสดงสถานะการทำงานของโปรแกรม เช่น การเชื่อมต่อของ socket, bind, listen, การหยุดการให้บริการ หรือ ค่าผิดพลาดต่างๆ ที่เกิดขึ้นระหว่างการทำงานของโปรแกรม

รูปแสดงลักษณะ Events Service Report แสดงรายละเอียดเมื่อมีการร้องขอทรัพยากรจากผู้ร้องขอ



ภาคผนวก ข

ข้อมูลทางอินเตอร์เน็ต "message/type"

ภาคผนวก ข อธิบาย โปรโตคอล HTTP/1.0 เพิ่มเติม ข้อมูลที่ให้บริการในอินเตอร์เน็ตจะต้องกำหนดแน่นอนในรูปแบบ "message/type" มีรูปแบบดังนี้

Media Type Name : message

Media Subtype name : http

Required parameters : none

Optional parameters : version, msgtype

เวอร์ชั่นของ HTTP ต้องใส่รวมมาด้วย เช่น HTTP/1.0 แต่หากไม่ได้กำหนดไว้ จะกำหนดไว้ในบรรทัดแรกของบรรทัด HTTP Header แล้ว Msgytpe ข้อความ Request หรือResponse หากไม่ได้กำหนดไว้ จะกำหนดไว้ในบรรทัดแรกของบรรทัด HTTP Header

Encoding consideration : ต้องเป็น "7bit" , "8bit", หรือ "binary" เท่านั้น

Security consideration : none

การนำเอกสารนี้ไปใช้

เอกสารนี้ เป็นข้อกำหนดตามมาตรฐาน HTTP/1.0 ไม่สามารถรองรับการทำงาน เวอร์ชันที่สูงกว่าได้เป็นข้อกำหนดกว้าง ๆ ไม่ได้เฉพาะเจาะจงหรือเป็นกฎเกณฑ์เสียทีเดียว ปกติจะคั่นคำสั่งของ HTTP ด้วยช่องว่าง และจบบรรทัดคำสั่งด้วยรหัส CR หรือ LF

ภาคผนวก ค
ความเกี่ยวข้องของ MIME

HTTP/1.0 ถูกนิยามมาจาก RFC 822 จุดประสงค์ของการใช้งานจดหมายทางอินเตอร์เน็ต (Multipurpose Internet Mail Extention: MIME) ที่ใช้ขนถ่ายข้อมูลได้หลายรูปแบบ และยังมี RFC 1521 อีกด้วย แต่ HTTP/1.0 จะแตกต่างจาก RFC 1521 คือ มีประสิทธิภาพมากกว่าและมีความเป็นอิสระมากขึ้นในการใช้ข้อมูล(Media Type) หัวข้อนี้จะอธิบายข้อแตกต่างของ HTTP/1.0 จาก RFC 1521

1. เปลี่ยนแปลงด้านรูปแบบกฎเกณฑ์

RFC 1521 ต้องการขนส่งข้อมูลโดย Internet Mail ใช้ Content-Type คือ text จะขบบรรทัดการแสดงผลด้วยเครื่องหมาย CRLF แต่ไม่มี CR และ LF แต่ว่า HTTP/1.0 มีพรอกซี หรือเกตเวย์ แปลความหมายการจบบรรทัดของ RFC 1521 ด้วย CRLF ซึ่งอาจจะก่อให้เกิดความยุ่งยากและต้องนำไปแสดงในส่วน content-encoding แทน

2. แปลงด้านรูปแบบของวันที่

HTTP/1.0 จะกำหนดรูปแบบของวันที่ตามมาตรฐาน GMT

3. การใช้ content-encoding

RFC 1521 ไม่ได้ใช้ content-encoding หากกรณีนี้เกิดขึ้นข้อมูลที่ พรอกซีและเกตเวย์จาก HTTP ไปยัง MIME ควรทำการแปลงข้อมูลไปยังอีกรูปแบบหนึ่งก่อน โดยไปรวมกับเฮดเดอร์ใน content-type

4. ไม่มี content-transfer-encoding

HTTP/1.0 ไม่มี content-transfer-encoding (CTE) เหมือนกับ RFC 1521

5. HTTP Header มีหลายรูปแบบ

ใน RFC 1521จะตัดเฮดเดอร์นี้ทิ้งไป แต่ใน HTTP/1.0 จะมี ชื่อเฮดเดอร์ ที่สื่อความหมาย

ภาคผนวก ง
คุณสมบัติเพิ่มเติม

ลักษณะการตอบสนองหรือการร้องขอในการทำงานระหว่าง เซิร์ฟเวอร์ กับ ไคลเอนต์ ถึงแม้จะมักไม่ค่อยได้ใช้ แต่ เซิร์ฟเวอร์ ก็ควรสามารถรองรับการทำงานได้

1. เมธอดในการร้องขอ

- PUT

การส่งข้อมูลไปยัง เซิร์ฟเวอร์ หากว่ามีข้อมูลอยู่ที่ เซิร์ฟเวอร์ อยู่แล้ว เมธอดนี้ก็จะทำการ เปลี่ยนแปลงข้อมูลให้ใหม่ขึ้น แต้ถ้าหากยังไม่มีการแก้ไขข้อมูล เซิร์ฟเวอร์ ก็จะสร้างข้อมูลขึ้นมาตามการร้องขอของ Request-URI

ความแตกต่างของการส่งข้อมูลแบบ POST กับ PUT คือ ผลที่เกิดขึ้นกับทั้ง 2 เมธอดนี้แตกต่างกัน หมายความว่า เมธอด POST จะส่งข้อมูลไปยัง เซิร์ฟเวอร์ เพื่อให้ เซิร์ฟเวอร์ ทำการประมวลผล หรือให้โปรแกรมใน เซิร์ฟเวอร์ รับไปทำงาน แต่ เมธอด PUT จะส่งข้อมูลไปเก็บไว้ที่เซิร์ฟเวอร์

- DELETE

การลบข้อมูลหรือทรัพยากรที่อยู่ใน เซิร์ฟเวอร์ โดยการสร้างคำร้องขอจาก Request-URI

- LINK

เพิ่มส่วนการเชื่อมโยงหลายมิติของข้อมูล โดยการสร้างคำร้องขอจาก Request-URI

- UNLINK

เคลื่อนย้ายส่วนการเชื่อมโยงหลายมิติของข้อมูล โดยการสร้างคำร้องขอจาก Request-URI

2. เฮดเดอร์

- Accept

เฮดเดอร์นี้ใช้แสดงรายการขอบเขตของข้อมูลที่สามารถยอมรับได้ โดย เซิร์ฟเวอร์ จะส่งการตอบสนองไปยัง ไคลเอนต์ เครื่องหมายดอกจันจะใช้แทนความหมายว่าเป็นชนิดของข้อมูลทุกรูปแบบ (* / *) และ"type/*" คือทุกรูปแบบ subtype client จะเป็นผู้ส่งการร้องขอว่าสามารถแสดงผลข้อมูลชนิดใดได้บ้าง

- Accept-Charset

ใช้แสดงรายการรูปแบบของตัวอักษร โดยปกติจะเป็นมาตรฐาน US-ASCII และ ISO-8859-1 เพื่อให้ ไคลเอนต์ แสดงผลข้อมูลได้ถูกต้อง

- Accept-Encoding

เฮดเดอร์นี้จะคล้ายกับ เฮดเดอร์ Accept แต่มีข้อจำกัดอยู่ที่ ต้องยอมรับค่าใน content-coding

- Accept-Language

คล้าย Accept แต่จำกัดที่ขอบเขตของภาษาในการตอบสนองข้อความ

- Content-Language

อธิบายลักษณะธรรมชาติของภาษาที่ใช้แสดงผลอาจจะมีลักษณะต่าง ๆ กัน

- LINK

อธิบายความสัมพันธ์ของข้อมูลในข้อความร้องขอ ในด้านโครงสร้างและการทำงาน การเชื่อมโยงมิติอาจมีมากกว่าหนึ่ง ข้อความร้องขอก็ได้

- MIME version

ใช้แสดงเวอร์ชันของ MIME protocol ที่ใช้สร้างข้อความตอบสนอง

- Retry-After

เป็นเฮดเดอร์ตอบสนอง 503 (service unavaiable) ใช้แสดงว่านานเท่าไรที่จะตัดการร้องขอจาก ไคลเอนต์ ค่าของเฮดเดอร์อาจจะเป็น HTTP-Date หรือจำนวนวินาที หลังจากที่เซิร์ฟเวอร์ ตอบสนองข้อความร้องขอ

- URI

เป็นรูปแบบชื่อ สถานที่ ข้อมูลหรืออื่นๆ ใช้บอกลักษณะของการเข้าถึงข้อมูลอาจจะเป็น www address ก็ได้

ประวัติผู้จัดทำโครงงาน

ชื่อ-สกุล นางสาวจริยา มุ่งคีมกลาง
วัน/เดือน/ปี เกิด 9 พฤศจิกายน 2521
ภูมิลำเนา 197/1 หมู่ 4 ต.แสนตอ อ.น้ำปาด จ.อุตรดิตถ์
ประวัติการศึกษา จบการศึกษาระดับมัธยมศึกษาตอนปลายจากโรงเรียนน้ำปาดชนูปถัมภ์
ปัจจุบันกำลังศึกษาอยู่คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมคอมพิวเตอร์
ชั้นปีที่ 4 มหาวิทยาลัยนเรศวร

ชื่อ-สกุล นางสาวจันทร์นิภา จันทร์แจ้ง
วัน/เดือน/ปี เกิด 13 มีนาคม 2521
ภูมิลำเนา 75/3 หมู่ 6 ต.สามง่าม อ.สามง่าม จ.พิจิตร
ประวัติการศึกษา จบการศึกษาระดับมัธยมศึกษาตอนปลายจากโรงเรียนสามง่ามชนูปถัมภ์
ปัจจุบันกำลังศึกษาอยู่คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมคอมพิวเตอร์
ชั้นปีที่ 4 มหาวิทยาลัยนเรศวร

ชื่อ-สกุล นายอานนท์ จันทร์เจ๊ก
วัน/เดือน/ปี เกิด 1 ตุลาคม 2521
ภูมิลำเนา 149/1 หมู่ 1 ต.ขุนฝาง อ.เมือง จ.อุตรดิตถ์
ประวัติการศึกษา จบการศึกษาระดับมัธยมศึกษาตอนปลายจากโรงเรียนอุตรดิตถ์วิทยา
ปัจจุบันกำลังศึกษาอยู่คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมคอมพิวเตอร์
ชั้นปีที่ 4 มหาวิทยาลัยนเรศวร