รายงานการวิจัย

การค้นหาข้อมูลจากหลายแหล่ง

Multiple Search Engine

โดย

เยาวดี เต็มธนาภัทร์

แหล่งทุน

ทุนอุดหนุนการวิจัยจากเงินรายได้ของสถาบัน ประจำปี 2542

สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ

พ.ศ. 2542

บทคัดย่อ

ชื่อโครงการ (ภาษาไทย) การค้นหาข้อมูลจากหลายแหล่ง
(ภาษาอังกฤษ) Multiple Search Engine
ชื่อผู้วิจัย นางสาวเยาวดี เต็มธนาภัทร์
หน่วยงานที่สังกัด ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ
คณะวิทยาศาสตร์ประยุกต์
หมายเลขโทรศัพท์ x 4626
ได้รับทุนอุดหนุนวิจัยประเภท ทุนอุดหนุนการวิจัยจากเงินรายได้ของสถาบัน
ประจำปี พ.ศ. 2542 จำนวนเงิน 40,000.- บาท (สี่หมื่นบาทถ้วน)

บทคัดย่อภาษาไทย

งานวิจัยนี้มีจุดประสงค์เพื่อพัฒนาตัวค้นหาข้อมูลจากหลายแหล่ง (Multiple Search Engine) เพื่อให้การค้นหาข้อมูลสำหรับผู้ใช้บนอินเทอร์เน็ตสะดวก มีโอกาสที่จะได้ข้อมูลครบถ้วน และตรงความต้องการมากขึ้นจากการค้นหาจากหลายแหล่งข้อมูลในเวลาเดียวกัน อีกทั้งช่วยให้ผู้ใช้ไม่จำเป็นต้องเรียนรู้การใช้งานหรือการถามคำถามจากแต่ละแหล่งซึ่งอาจมีความแตกต่างกัน ในงานวิจัยนี้ได้เลือกใช้แหล่งข้อมูลจาก Search Engines 3 แหล่ง คือ AltaVista, Google และ Lycos ซึ่งเป็น Search Engine ที่ได้รับความนิยม

จากผลการดำเนินการ สามารถสร้างเครื่องมือการค้นหาจากหลายแหล่ง โดยส่งคำค้นของผู้ใช้แปลงให้เหมาะสมกับ Search Engine แต่ละตัว เมื่อได้รับคำตอบจาก Search Engine ผลลัพธ์ที่ได้จะถูกแปลงให้อยู่ในรูปแบบเดียวกัน และทยอยรวมกันตามความรวดเร็วของการตอบของ Search Engine ก่อนส่งคำตอบกลับไปยังผู้ใช้ และเพื่อรองรับการเปลี่ยนแปลงในเรื่องรูปแบบการค้นและการคืนผลลัพธ์ของแหล่งต้นตอ ในงานวิจัยนี้ใช้การสร้าง Profile ในรูปของ Regular Expression ช่วยเพิ่มความยืดหยุ่นในการการแก้ไขในภายหลัง

บทคัดย่อภาษาอังกฤษ (Abstract)

Project Name (Thai)	การค้นหาข้อมูลจากหลายแหล่ง
(English)	Multiple Search Engine
Researcher Name	Yaowadee Temtanapat
Affiliation	Department of Computer and Information Science
	Faculty of Applied Science
Phone	x 4626

Grant: Research Funding from King Mongkut's Institute of Technology North Bangkok's Fiscal Budget Year 1999: 40,000.- Bahts (Forty Thousand Bahts)

Abstract

This project aims to develop a multiple search engine to ease an Internet user in word searching on many general-purposed search engines at the same time. The system helps to provide a wider coverage of results and a unified interface in searching. The project chooses 3 main popular search engines: AltaVista, Google and Lycos as its main sources.

A result of the project is a metasearch tool that works as follows: first it accepts a user's request in a form of searching terms. It transforms this searching term into a format that will be suitable to a specific search engine. Then submit these new appropriate terms for searching to 3 search engines at the same time. It waits for results. Any results from the engines are gradually merged and reorganized to form result pages to the user. In addition, since the source search engine may change its format both in term of queries and results, the project chooses to create a profile of regular expressions that can be easily changed and configured.

กิตติกรรมประกาศ

(Acknowledgement)

ผู้วิจัยขอขอบคุณ สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ ที่ให้การสนับสนุนทุนวิจัยเพื่อการพัฒนางานในโครงการนี้ นอกจากนี้ขอขอบคุณ นายทศพล อัญชลีชไมกร ผู้ช่วยวิจัยที่ช่วยให้โครงการสำเร็จลุล่วงไปได้ด้วยดี

เยาวดี เต็มธนาภัทร์

เมษายน 2550

(ด้านคอมพิวเตอร์และสารสนเทศ)

สารบัญเรื่อง (Table of Contents)

หน้า

สารบัญตาราง (List of Tables)

หน้า

สารบัญภาพ (List of Illustrations)

หน้า

สารบัญภาพ (List of Illustrations)

หน้า

บทที่ 1

บทนำ (Introduction)

ความสำคัญและที่มาของปัญหา

ในปัจจุบันเครือข่ายใยแมงมุม (World Wide Web) เป็นที่แพร่หลายและได้รับความนิยมทั้งในแง่เพื่อการให้บริการ การดำเนินการธุรกิจ การประชาสัมพันธ์ และเป็นแหล่งข้อมูลและสารสนเทศขนาดใหญ่ การค้นหาข้อมูลเพื่อให้ได้ข้อมูลที่ต้องการเป็นเรื่องจำเป็นและมีความสำคัญ อย่างไรก็ตามข้อมูลที่มีอยู่ปริมาณมาก และหลากหลายในเว็บ จำเป็นจะต้องเข้าถึงโดยการค้นหาจากแหล่งข้อมูลหรือตัวค้นหาที่เป็นที่รู้จักกันดีในนามของ Search Engine การพัฒนา Search Engine เองประกอบด้วยขั้นตอน เทคโนโลยีและความรู้จากหลายด้าน การจะพัฒนา Search Engine จึงใช้เวลาและการลงทุนมาก จึงยากที่จะพัฒนาให้แข่งขันได้กับ Search Engine ที่มีอยู่ในปัจจุบัน อีกทั้ง Search Engine ก็มีให้เลือกใช้อยู่หลายตัว แต่ละตัวมีความสามารถและการให้ลำดับความเหมือนในการค้นที่แตกต่างกันออกไป

แนวทางหนึ่งที่จะช่วยในการค้นหา และเป็นการพัฒนาต่อยอดจาก Search Engine ทั่วไปคือการพยายามรวบรวมความสามารถที่แตกต่างกันของ Search Engine ที่มีอยู่มาสร้างตัวค้นหาในรูปแบบเดียว (Unified Interface) ครอบเหนือ Search Engine หลายแหล่งหรือที่เรียกกันอีกอย่างว่า MetaSearch ผลที่ได้คือ ตัวค้นหาที่มีความสามารถเหนือจากการค้นหาจากแหล่งเดียว ซึ่งทำให้ผู้ใช้สามารถที่จะใช้บริการของหลาย Search Engine ได้ในเวลาเดียวกัน โดยไม่ต้องศึกษาถึงการใช้ Search Engine แต่ละตัวหรือยึดติดกับการใช้ตัวใดตัวหนึ่ง

วัตถุประสงค์ของการวิจัย

เพื่อพัฒนาตัวค้นหาข้อมูลจากหลายแหล่ง อันจะเป็นประโยชน์ในการค้นหาข้อมูลสำหรับผู้ใช้ทั่วไป

ประโยชน์ที่คาดว่าจะได้รับ

- พัฒนาตัวการค้นหาเพื่ออำนวยความสะดวกในการค้นหาข้อมูลจาก Web โดยใช้ Uniform Interface ในการค้นหาจากแหล่งสืบค้นหลายแหล่ง
- พัฒนาโครงสร้างพื้นฐานการวิจัยสำหรับการทำงานด้าน Information Integration และการจัดการฐานข้อมูลแบบ non-structural

ขอบเขตของการวิจัย

โครงงานนี้จะสร้าง metasearch ขึ้นเพื่อเป็น Uniform Interface สำหรับการสืบค้นบนอินเทอร์เน็ตโดยมี Search Engine ทั่วไปที่มีประสิทธิภาพในการค้นหาที่ดีเป็นแหล่งการสืบค้น โดยสามารถค้นหาคำในรูปแบบของ “หรือ” และ “ทุกคำ (exact words)” และค้นหาภาษาไทยได้

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง (Theory and Related Works)

ในการพัฒนา MetaSearch จำเป็นที่จะต้องอาศัยความรู้พื้นฐานที่เกี่ยวข้องหลายประการ อาทิ ความรู้เบื้องต้นเกี่ยวกับการค้นหา เทคโนโลยีด้าน Client-Server, WWW, Network Programming, CGI, Caching, Regular Expression รวมถึงเทคโนโลยีการจัดการระบบฐานข้อมูล ดังนั้นในส่วนนี้จะนำเสนอความรู้เบื้องต้นเกี่ยวกับเทคโนโลยีดังกล่าว

ความรู้เบื้องต้นเกี่ยวกับการค้นหา

การสืบค้นสารสนเทศ (Information Retrieval)

การสืบค้นสารสนเทศ [12] เป็นเทคนิคที่ใช้เพื่อการค้นหาเอกสารที่เกี่ยวเนื่อง หรือเป็นประเด็นสำคัญสำหรับผู้ค้นจากเอกสารที่เก็บรวบรวมไว้ โดยเอกสารที่เก็บรวบรวมจะต้องผ่านกระบวนการและถูกจัดให้อยู่ในรูปแบบที่สะดวกและช่วยให้การค้นหาเป็นไปอย่างมีประสิทธิภาพ

โดยทั่วไปเนื้อหาในเอกสารจะถูกเก็บโดย represent ในรูปคำค้น คำค้นบางคำในภาษาอังกฤษเช่น a, an, the, of, is, in หรือคำภาษาไทยเช่น นะ คะ ครับ และคำใกล้เคียงอื่น ๆ โดยเราพบว่าคำเหล่านี้เป็นคำที่ไม่มีความหมาย (semantic) และเรียกคำในลักษณะนี้ว่า *Stop Word* และโดยทั่วไปจะไม่ถูกเก็บเป็นคำค้นเพื่อ represent เอกสาร สำหรับคำนอกจาก Stop Words สามารถใช้ represent เอกสารได้ นอกจากนี้คำจำนวนมากมีหลากหลายรูป เช่น develop, development, developing, developed อาจถูก represent ด้วยคำ develop ซึ่งการจัดการกับความหลากหลายรูปของคำให้อยู่ในรูปใดรูปหนึ่งเรียกว่าการทำ Stemming หลังจากกำจัดคำ stop และ stemming แล้วสามารถ represent ในระดับแนวคิดได้โดยใช้ *เวคเตอร์ของ n-terms* เมื่อ n เป็นจำนวนคำที่แตกต่างที่มีทั้งหมดในกลุ่มของเอกสารที่เก็บรวบรวมไว้

สมมติว่าเอกสาร d ถูก represented ด้วยเวคเตอร์ (d_1, …, d_i, …, d_n) เมื่อ d_i คือน้ำหนักหรือตัวเลขแสดงความสำคัญของ term ที่ i^{th} ในเอกสาร d โดยส่วนใหญ่ค่าใน entry ของเวคเตอร์จะเป็นศูนย์ เพราะ term ทั้งหมดที่มีโดยส่วนใหญ่ไม่ปรากฏในเอกสาร หากเอกสารใดมี term ใดปรากฏน้ำหนักหรือตัวเลขแสดงความสำคัญจึงจะมีค่า และค่านี้อาจถูกกำหนดโดยใช้ความถี่ที่ปรากฏของ term นั้นในเอกสาร (term frequency) หรือ อาจถูกกำหนดจากค่าความถี่ที่ term นั้นปรากฏในเอกสารต่าง ๆ ที่เก็บรวบรวมไว้ (document frequency)

การค้นหาเอกสารทำได้โดยกำหนดคำค้นในรูป text ซึ่งจะถูกแปลงให้กลายเป็นเวคเตอร์ n-มิติ กล่าวคือคำค้นจะผ่านกระบวนการกำจัด stop words และ stemming จากนั้นคำค้นที่ถูกแปลงแล้วจะถูกนำไปเปรียบเทียบกับ term โดยอาศัยฟังก์ชันการเปรียบเทียบที่เรียกว่า Similarity Function เพื่อวัดความใกล้เคียงของคำค้นกับ term ของเอกสาร จากนั้นก็จะนำเอกสารที่ปรากฏคำค้นนั้นมาจัดลำดับความสำคัญตามลำดับความใกล้เคียงเพื่อส่งเป็นคำตอบ

การวัดประสิทธิภาพของการสืบค้นสารสนเทศ มักใช้ค่าความเที่ยง (Precision) (สมการ (1)) และค่าเรียกกลับ (Recall) ซึ่งถูกกำหนดไว้ดังสมการที่ (1) และ (2) ตามลำดับ

$$\text{Precision} = \frac{\text{the number of retreived relevant documents}}{\text{the number of relevant documents}} \qquad (1)$$

$$\text{Recall} = \frac{\text{the number of retreived relevant documents}}{\text{the number of retrieved documents}} \qquad (2)$$

การสืบค้นสารสนเทศที่มีประสิทธิภาพ ควรจะต้องเลือกเอกสารที่เกี่ยวเนื่องทั้งหมดและไม่เลือกเอกสารที่ไม่เกี่ยวเนื่องอื่นเลย กล่าวคือค่าของทั้งความเที่ยงและค่าเรียกกลับเป็น 1 แต่ในทางปฏิบัติทั่วไปแล้วไม่มีทางเป็นไปได้ เนื่องจากความเกี่ยวเนื่องนั้นเป็นเรื่องที่ขึ้นกับบุคคลและเวลา ดังนั้นความเกี่ยวเนื่องจึงอาจแปรเปลี่ยนและยากในการวัด

Engine สำหรับการค้นหา

ในการค้นหาบนอินเทอร์เน็ต อาจแบ่งเครื่องมือ [15] ที่ใช้ในการค้นหาได้เป็น 2 ประเภทหลักคือ Search Engine ทั่วไป (General Purpose Search Engine) และ MetaSearch Engine

Search Engine ทั่วไป

Search Engine ทั่วไป มีจุดประสงค์หลักเพื่อให้สามารถค้นหาข้อมูลต่าง ๆ บนเว็บเพจได้ โดยทั่วไป Search Engine ประเภทนี้จะเก็บข้อมูลในรูปของข้อความ (Text) โดยมีการสร้างดัชนีสำหรับคำ (Term) เพื่อใช้ช่วยในการค้นหา ตัวอย่างของ Search Engine เหล่านี้ได้แก่ AltaVista [3], HotBot [11], Lycos [14] เป็นต้น

หลักการทำงานของ Search Engine ทั่วไปอาจแบ่งได้เป็น 2 ขั้นตอนหลักคือ ขั้นการรวบรวมและสร้างดัชนี และขั้นการค้นหา

ขั้นการรวบรวมและสร้างดัชนี โดยทั่วไป Search Engine จะใช้โปรแกรมที่เรียกว่า Web Robots หรือ Web Crawler เพื่อทำหน้าที่เก็บรวบรวมเอกสารบนหน้าเว็บ Web Robot จะท่องไปยัง Web Server ต่าง ๆ เพื่อรวบรวมเอกสารบนเว็บ มาสร้างเป็นดัชนีของคำค้นโดยอัตโนมัติ ดัชนีของคำค้นนี้โดยทั่วไปอาจใช้ดัชนีในรูปแบบที่เรียกว่า Inverted Index โดยจะสร้างลิสต์ในรูปของ Hash table หรือ binary tree ระหว่างคำค้นกับตำแหน่งที่เชื่อมไปยังเอกสารที่พบคำค้นนั้น

ขั้นตอนในการค้นหา แต่ละ Search Engine จะมีเทคนิคที่ใช้ในการค้นหาและเรียงลำดับคำตอบจากการค้นหาที่แตกต่างกันออกไป โดยทั่วไปใช้เทคนิคในลักษณะเดียวกันการสืบค้นสารสนเทศ แต่มีวิธีการกำหนดน้ำหนักหรือค่าความสำคัญแตกต่างกันไปตามแต่ Search Engine

อย่างไรก็ตามการที่ Search Engine ค้นหาข้อมูลจากหน้าเว็บซึ่งเป็นเอกสาร HTML ที่มี Tag Search Engine อาจใช้ Tag เช่น Meta, Title, Head เป็นส่วนหนึ่งในการกำหนดน้ำหนักความสำคัญของเอกสาร

นั้นกับ term ตัวอย่างเช่นอาจพิจารณาว่าคำที่ปรากฏใน Title, Head มีความสำคัญมากกว่าคำที่ปรากฏในตัวเอกสาร เป็นต้น

MetaSearch Engine

MetaSearch Engine หรือเรียกกันอย่างย่อว่า *MetaSearch* เป็นระบบซึ่งพยายามสร้างวิธีการเข้าถึงข้อมูลจากหลายแหล่งข้อมูล โดยใช้รูปแบบการเข้าถึงเพียงรูปแบบเดียว (Unified Access Interface) โดยทั่วไป MetaSearch จะไม่เก็บรวบรวมเอกสาร หรือเก็บดัชนีของข้อมูลด้วยตนเอง แต่อาจมีบ้างที่จำเป็นต้องเก็บข้อมูลบางส่วนเพื่อให้บริการการค้นหามีประสิทธิภาพขึ้น หลักการทำงานของ MetaSearch เริ่มต้นจากการรับคำค้น หรือคิวรี (Query) จากผู้ใช้ จากนั้นจัดรูปแบบของคิวรีที่ได้รับนั้นใหม่ (reformatting) เพื่อให้เหมาะกับ Search Engine ซึ่งจะเป็นแหล่งข้อมูลการค้นหาที่แท้จริง หลังจากส่งคิวรีที่ปรับแก้ซึ่งเหมาะสมกับ Search Engine เฉพาะตัวแล้วนั้น เมื่อได้รับคำตอบจาก Search Engine ก็จะทำการ organize และจัดลำดับข้อมูลที่ได้รับให้อยู่ในรูปแบบคำตอบที่ MetaSearch กำหนดไว้ภายใน

การสร้าง MetaSearch ไม่ใช่การทำงานซ้ำซ้อนกับสิ่งที่ Search Engine ทั่วไปดำเนินการไว้ แต่มีหลายเหตุผลในการสร้าง อาทิ

- *เพื่อช่วยเพิ่มความครอบคลุมในการค้นหา* เนื่องจากจำนวนเอกสารบนเว็บมีปริมาณเพิ่มขึ้นเรื่อย ๆ และเป็นจำนวนมาก Search Engine แต่ละตัวอาจมีความสามารถในการค้นหาที่แตกต่างกันออกไป เช่น Yahoo อาจสามารถค้นหาในเรื่องของประเภท เพราะมีการจัดประเภทของเอกสารได้ดี ในขณะที่การค้นหาคำค้นทั่วไปอาจทำได้ดีกว่าโดย AltaVista เป็นต้น ดังนั้นการรวบรวมผลลัพธ์ในการค้นหาจากหลายแหล่งจึงน่าจะมีโอกาสที่จะให้ผลลัพธ์ซึ่งเป็นที่พอใจได้มากกว่าการค้นจากแหล่งเดียว
- *ช่วยเพิ่มความสะดวกในการค้นหาจากหลาย Search Engine พร้อมกัน* ในบางครั้งความต้องการข้อมูลของผู้ใช้อาจได้มาจากการรวบรวมคำตอบจากหลายแหล่ง การที่ผู้ใช้ต้องดำเนินการค้นหาจากแหล่งต่าง ๆ ด้วยตนเอง จำเป็นจะต้องทราบถึง URL ของที่ตั้งของข้อมูล รู้จักวิธีการทำคิวรีบนแหล่งข้อมูลเหล่านั้น และจำเป็นต้องติดต่อกับหลายแหล่ง และรวบรวมข้อมูลจากหลายแหล่งนั้นด้วยตนเอง ไม่มีตัวช่วยในการจัดข้อมูลทั้งหมดเข้าด้วยกัน
- *เพื่อช่วยเพิ่มประสิทธิภาพในการสืบค้น* การสืบค้นของงานแต่ละประเภทอาจตั้งอยู่บนพื้นฐานความต้องการที่แตกต่างกันออกไป ไม่จำเป็นเสมอไปว่าการสืบค้นบน Search Engine ทั่วไปจะให้ผลลัพธ์ที่ดีที่สุด อย่างไรก็ตาม หากผู้ใช้สามารถสร้างทางเลือกในการค้นหา ก็จะช่วยให้สามารถสืบค้นได้อย่างมีประสิทธิภาพมากยิ่งขึ้น ตัวอย่างเช่น MetaSearch อาจช่วยค้นหาได้จากแหล่งทั่วไปและแหล่งเฉพาะด้านอาทิ Help Desk Search Engine (Search Engine ที่

รวบรวมคำถาม-คำตอบในการซ่อมอุปกรณ์) ในคิวรีที่ผู้ใช้ต้องการแก้ปัญหาการซ่อม Help Desk Search Engine หากมีทางเลือกที่สะดวกสำหรับผู้ใช้ในการกำหนดแหล่งสืบค้นก็จะทำให้สืบค้นได้อย่างมีประสิทธิภาพ และอาจให้คำตอบที่เหมาะสมมากกว่า Search Engine ทั่วไป

ขั้นตอนการรวมข้อมูลหลายแหล่งเข้าด้วยกัน (Integration หรือ Merging)

การรวบข้อมูลผลลัพธ์จากหลายแหล่ง Search Engine เข้าด้วยกัน หรือการทำ Integration หรือ การ Merging ประกอบด้วยขั้นตอนหลักคือ

- การแปลงและคัดเลือกข้อมูลเพื่อการรวม
- การจัด organize และเรียงข้อมูลให้อยู่ในรูปแบบคำตอบที่เหมาะสม

ในการแปลงข้อมูลให้มีรูปแบบที่สอดคล้องกัน เนื่องจากข้อมูลจากหลายแหล่งอาจมี schema ของข้อมูลที่กำหนดไว้แตกต่างกันออกไป ในบางแหล่งข้อมูลอาจมีชนิดข้อมูลมากกว่าบางแหล่ง บางแหล่งอาจมีชนิดข้อมูลที่แตกต่างกันไปเลย การแปลงข้อมูลเพื่อให้มีรูปแบบที่เป็น uniform จึงอาจกำหนด Unified Schema สำหรับข้อมูลที่ต้องการรวม ในกรณีนี้อาจกำหนด schema โดยคัดเลือกข้อมูลเพื่อการรวบรวม โดยอาจกำหนด Unified Schema แบบ

1. การรวมแบบให้ข้อมูลมากที่สุด : เนื่องจากชนิดข้อมูลอาจมีความแตกต่างกัน อาจเลือกการรวมโดยให้คำตอบที่แสดงรายละเอียดให้มากที่สุดที่เป็นไปได้ ในกรณีที่แหล่งข้อมูลอื่นไม่มีข้อมูลเหล่านั้นให้เว้นว่างชนิดข้อมูลเหล่านั้นไป ในกรณีที่ข้อมูลมีความแตกต่างกันก็รวบรวมข้อมูลที่ต่างนั้นไว้เป็นคำตอบด้วย
2. การรวมแบบให้ข้อมูลร่วมเท่านั้น : ในกรณีนี้พิจารณาเพียงรวมข้อมูลร่วมที่มีเหมือนกันในทุกแหล่งข้อมูล และนำเสนอผลลัพธ์เพียงข้อมูลที่มีร่วมกันเท่านั้น
3. การรวมแบบผสม : ในกรณีนี้เป็นการรวมแบบผสมของทั้ง 2 แบบข้างต้น ซึ่งพิจารณารวมข้อมูลโดยเลือกทั้งที่เป็นข้อมูลร่วม และบางส่วนของข้อมูลที่แตกต่างกัน

สำหรับในกรณี Search Engine ข้อมูลที่มีลักษณะร่วมเหมือนกันในคำตอบของ Search Engine คือ ส่วน anchor หรือ hyperlink ที่อ้างอิงไปยังหน้าเว็บของเอกสาร คำขึ้นต้น คำอธิบายโดยย่อเกี่ยวกับเอกสารของคำตอบนั้น ในบาง Search Engine อาจมีรายละเอียดอื่น เช่น Ranking value ของเอกสาร (เช่นใน AltaVista) ซึ่งอาจไม่มีความสำคัญนักในแง่ของผู้ใช้ ดังนั้นในกรณีนี้การรวมจึงใช้แบบให้ข้อมูลร่วมเท่านั้น

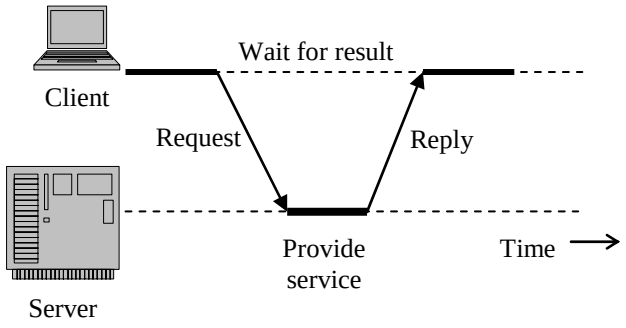
หลังจากคัดเลือกข้อมูลที่ต้องการจากแหล่งต่าง ๆ ได้แล้ว ขั้นถัดไปคือการนำข้อมูลเหล่านั้นมาจัดรูปแบบใหม่ให้เหมาะสมกับการสร้างเป็นคำตอบ ในกรณีที่คำตอบที่ได้เหมือนกันมากกว่าหนึ่งแหล่งต้องกำจัดความซ้ำซ้อนก่อนการรวม นอกจากนี้เนื่องจากลำดับความเหมือน (similarity) ของคำตอบว่าสอดคล้องกับคำค้นมากเพียงใด ในแต่ละ Search Engine มีการจัดลำดับที่แตกต่างกันไป ทำให้ต้องกำหนดลำดับของผลลัพธ์เมื่อรวมเข้าเป็นผลลัพธ์ของ MetaSearch อย่างไรก็ตาม Search Engine อาจไม่ให้ข้อมูลความเหมือนซึ่งทำให้ตัดสินไม่ได้ว่าลำดับความเหมือนของคำตอบในระดับรวมควรเป็น

อย่างไร วิธีการอย่างหนึ่งที่อาจใช้ในการรวมคือการถ่วงน้ำหนักความสำคัญของแหล่งต้นทางข้อมูลว่าแหล่งใดน่าจะให้คำตอบที่ดีที่สุด ร่วมกับลำดับที่ปรากฏของคำตอบ

เทคโนโลยี Client/Server

ทั่วไป Client/Server [23] หมายถึงสถาปัตยกรรมที่จัดแบ่งการทำงานออกเป็นผู้ขอบริการ หรือ Client กับผู้ให้บริการ หรือ Server ในส่วนนี้จะอธิบายถึงโมเดลของ Client/Server

Processes ในโมเดลพื้นฐานของ Client/Server ถูกแบ่งออกเป็น 2 กลุ่มหลัก โดยที่ Server Process ถูก implemented เพื่อให้บริการเฉพาะบางอย่าง ตัวอย่างของบริการเหล่านี้เช่น file system service หรือ database service สำหรับ Client เป็น process ซึ่งร้องขอบริการจาก server โดยการส่ง request จากนั้นรอคำตอบหรือ reply จาก server ทำให้เรียกพฤติกรรมนี้ว่า Request-Reply Behavior ซึ่งอาจแสดงปฏิสัมพันธ์ระหว่าง Client และ Server ได้ดังรูปที่ 2.1



รูปที่ 2.1 ปฏิสัมพันธ์ระหว่าง Client และ Server

การสื่อสารระหว่าง client และ server ทั่วไปจะมีการกำหนดรูปแบบการ request และ reply ที่เรียกว่าโปรโตคอล (Protocol) โดยอาจใช้โปรโตคอลอย่างง่ายอาทิ *Connectionless Protocol* ในกรณีที่เครือข่ายมีความเชื่อถือได้สูงอย่างใน LAN ในกรณีนี้เมื่อ client ร้องขอบริการ มันจะสร้าง package ของ message สำหรับ server โดยระบุบริการที่ต้องการ และข้อมูลนำเข้าที่จำเป็นสำหรับบริการนั้น แล้วจึงส่งไปยัง server ในทางกลับกัน server ซึ่งเฝ้ารอรับคำร้องขอบริการอยู่ตลอดเวลา เมื่อได้รับคำร้องก็จะประมวลผลบริการ จากนั้นสร้าง package ของคำตอบเป็น reply message เพื่อส่งกลับไปยัง client

จะเห็นว่า Connectionless Protocol มีข้อดีในแง่ของประสิทธิภาพ ตราบเท่าที่ message ไม่สูญหายหรือเสียหาย อย่างไรก็ดีในระบบเครือข่าย การกำหนด Protocol ที่ทนต่อความผิดพลาดจากการส่งผ่านนั้นไม่ง่าย client อาจแก้ปัญหานี้ได้โดยส่งคำร้องซ้ำหากไม่ได้รับคำตอบกลับ แต่ก็ไม่ง่ายนักสำหรับ client ที่จะทราบได้ว่า คำร้องขอนั้นสูญหายระหว่างทางหรือไม่ ในบางกรณีที่ไม่อาจยอมรับที่การบริการนั้นจะถูกดำเนินการมากกว่าหนึ่งครั้ง เช่นในกรณีการถอนเงินหากดำเนินการซ้ำสองครั้งจะทำให้ผลลัพธ์ของยอดเงินคงเหลือผิดพลาด ดังนั้นทางเลือกอีกทางหนึ่งที่ทำได้คือใช้ Connection-Oriented Protocol ซึ่งอาจให้ประสิทธิภาพการทำงานที่ด้อยกว่า เพราะในโปรโตคอลนี้เมื่อ client ต้องการร้องขอ

บริการ เริ่มต้น client จะต้องสร้างการเชื่อมต่อไปยัง server ก่อนที่จะมีการร้องขอ สำหรับ server ก็จะใช้ช่องทางนี้ในการตอบกลับ การสร้างการเชื่อมต่อและยกเลิกการเชื่อมต่อโดยทั่วไปจะมี overhead ในแง่ของเวลาค่อนข้างมาก

WWW (World Wide Web)

อาจมองได้ว่า WWW คือระบบแบบกระจายที่ประกอบด้วย clients และ servers จำนวนมหาศาลเพื่อการเข้าถึงเอกสาร แต่ละ server เก็บรวบรวมเอกสาร โดยที่แต่ละเอกสารถูกเก็บไว้ในรูปของไฟล์ เมื่อ server ได้รับคำร้องขอก็จะดึงและส่งเอกสารนั้นไปยัง client

เอกสารสำหรับเว็บ มักจะถูกเขียนด้วยภาษา Hypertext Markup Language (HTML) ตัว Markup หรือ Tag ของคำสงวนจะเขียนโดยกำกับด้วยเครื่องหมาย “<” และ “>” โดย Tag จะเป็นตัวกำหนดรูปแบบการนำเสนอหรือโครงสร้างการนำเสนอเอกสารนั้น ข้อความที่ต้องการจัดรูปแบบจะถูกกำกับระหว่าง Tag เปิด <…> และ Tag ปิด </…> เช่น `<H1>Welcome</H1>` เป็นการกำหนดรูปแบบให้นำเสนอข้อความ Welcome ด้วยตัวขนาดใหญ่ โดยทั่วไปเอกสาร HTML ประกอบด้วยส่วน Head และ Body ตัวอย่างของเอกสาร HTML อย่างง่ายแสดงในรูปที่ 2.2

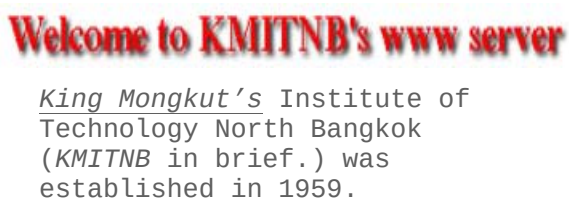
```
<HTML>
   <HEAD>
      <TITLE>My Page</TITLE>
   </HEAD>
   <BODY>
      <H1>Welcome</H1>
   </BODY>
</HTML>
```

รูปที่ 2.2 ตัวอย่างเอกสาร HTML

เอกสาร HTML อาจมีเพียงข้อความอย่างตัวอย่างในรูปที่ 2.2 หรืออาจบรรจุ image หรือ link ไปยัง site อื่นก็ได้ดังรูปที่ 2.3 (a) และเมื่อแสดงบน Browser ได้ดังรูปที่ 2.3 (b) โดย <img …> เป็น tag อ้างถึงรูปภาพที่มีแหล่งต้นทางที่ค่าที่กำหนดใน attribute src และ tag <a …> เป็น hyperlink tag ซึ่งจะอ้างไปยังแหล่งข้อมูลอื่นที่กำหนดไว้ใน attribute href

```
<img src="welcome-bar.gif" border=0 alt="">
<br>
<A HREF="http://www.kmitnb.ac.th/KingMongkut/kingmke.html">
<i>King Mongkut</i></A>'s Institute of Technology North Bangkok
(<i>KMITNB</i> in brief.) was established in 1959.
```

(a)



(b)

รูปที่ 2.3 ตัวอย่างเอกสาร HTML ที่มี image และ link ไปยังเอกสารอื่น

หรืออาจอยู่ในรูปของฟอร์มของ HTML [6] ที่ผู้ใช้สามารถส่งข้อมูลเข้าเพื่อให้ดำเนินบริการนั้นก็ได้ ดังโค้ดตัวอย่างต่อไปนี้ ซึ่งเมื่อนำเสนอบน Browser จะได้ผลดังในรูปที่ 2.4

```
<HTML>
<HEAD>
  <TITLE>Subscription Form</TITLE>
</HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF">
    <CENTER><H3> Subscription Form</H3></CENTER>
    <FORM ACTION="/cgi-bin/subscriber.cgi?" METHOD="POST">
      <P>
       <LABEL FOR="firstname">First name: </LABEL>
       <INPUT TYPE="text" NAME="firstname" SIZE="20"><BR>
       <LABEL FOR="lastname">Last name: </LABEL>
       <INPUT TYPE="text" NAME="lastname"><BR>
       <LABEL FOR="email">email: </LABEL>
       <INPUT TYPE="text" NAME="email"><BR>
       <INPUT TYPE="radio" NAME="sex" VALUE="Male"> Male<BR>
       <INPUT TYPE="radio" NAME="sex" VALUE="Female"> Female<BR>
       <INPUT TYPE="submit" value="send">
       <INPUT TYPE="reset">
      </P>
    </FORM>
</BODY>
</HTML>
```

โดยที่ Form:

ACTION คือชื่อโปรแกรมบน Server ที่จะเป็นตัวให้บริการตามคำร้องขอจากฟอร์ม โดยชื่อตัวบริการประกอบด้วย URL ที่จะชี้ไปยังโปรแกรมบน Server

METHOD คือคำสั่งในการกำหนดวิธีการร้องขอภายใต้โปรโตคอล HTTP คำสั่งที่ใช้ทั่วไปได้แก่ GET, HEAD และ POST

INPUT คือส่วนพารามิเตอร์นำเข้าเพื่อให้สามารถทำงานตามบริการที่ร้องขอได้ ตัวอย่างรูปแบบของ tag INPUT เป็นดังนี้

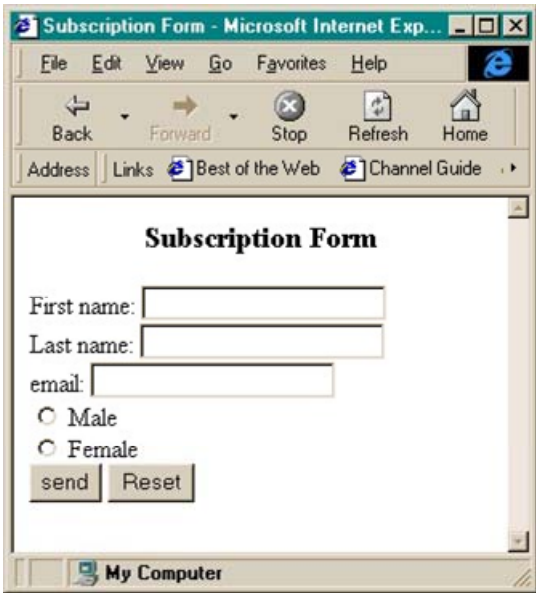
<INPUT TYPE=*[text | password | radio | checkbox | submit | reset | hidden]* NAME=*[string]* VALUE=*[string]*>

โดยที่ ชนิดของ INPUT (หรือ TYPE) อาจเป็น

- Text ข้อมูลชนิดข้อความ
- Password ข้อมูลชนิดรหัสผ่าน ซึ่งเมื่อคีย์ข้อมูลจะไม่ปรากฏเป็นข้อความที่อ่านได้
- Radio ข้อมูลชนิดตัวเลือก แบบที่เลือกได้ไม่เกิน 1 รายการ
- Checkbox ข้อมูลชนิดตัวเลือก แบบที่เลือกได้มากกว่า 1 รายการ
- Hidden ข้อมูลนำเข้าที่ถูกซ่อนจากผู้ใช้
- Submit ปุ่มกดเพื่อยืนยันการส่งการร้องขอ
- Reset ปุ่มกดเพื่อยกเลิกรายการที่ป้อนเข้าทั้งหมด เพื่อให้กลับไปกรอกรายการใหม่

สำหรับ NAME คือชื่อของพารามิเตอร์เข้า และ VALUE คือค่าเริ่มต้นที่กำหนดให้กับพารามิเตอร์ NAME นั้น

LABEL คือ tag เพื่อแสดงข้อความ label ใช้ในกรณีที่ไม่มี implicit label เพื่อให้คำอธิบายหรือให้รายละเอียดกับผู้ใช้



รูปที่ 2.4 ตัวอย่างฟอร์มเมื่อนำเสนอจาก Web Browser

ในการอ้างถึงเอกสาร WWW ใช้การอ้างอิงผ่าน Uniform Resource Locator (URL) ซึ่งเป็นตัวระบุที่ตั้งของเอกสาร โดยทั่วไปจะประกอบจากชื่อของ DNS, path ที่เป็นที่ตั้งและชื่อของไฟล์เอกสารนั้น นอกจากนี้ URL ยังอาจระบุถึง Application-Level Protocol ที่ใช้ในการส่งผ่านเอกสารด้วย โปรโตคอลทั่วไปที่ใช้คือ *Hypertext Transfer Protocol (HTTP)* ซึ่งเป็นโปรโตคอลอย่างง่าย โดยเริ่มจากการเปิดการเชื่อมต่อของ client มายัง server ส่งคำร้องขอ รอรับคำตอบ เมื่อ server ส่งคำตอบกลับแล้วก็จะปิดการ

เชื่อมต่อลง ทำให้ไม่สามารถจดจำสถานะของการทำงานก่อนหน้ากับ client นั้นได้ หรือที่เรียกว่า *Stateless*

ใน HTTP มี method สำหรับการร้องขอสรุปได้ดังตารางที่ 2.1 และในการตอบกลับมีรหัสการตอบกลับ สรุปได้ดังตารางที่ 2.2

ตารางที่ 2.1 แสดง Method ที่สนับสนุนใน HTTP

Method	**คำอธิบาย**
HEAD	เป็นคำร้องขอ header ของเอกสาร
GET	เป็นคำร้องขอตัวเอกสาร
PUT	เป็นคำร้องขอให้เก็บเอกสาร
POST	เป็นคำร้องขอ ที่มีการส่งข้อมูลไปกับเอกสารด้วย เช่นการส่ง submit ข้อมูลจากฟอร์ม
DELETE	เป็นคำร้องขอให้ลบเอกสาร

ตารางที่ 2.2 แสดงรหัสตอบกลับใน HTTP

รหัส	**คำอธิบาย**
1xx	Informational แสดงถึงกำลังดำเนินการตามคำร้องขอ
2xx	Success แสดงว่าผลการทำงานเป็นปกติ
3xx	Redirection แสดงการส่งต่อไปยังที่ตั้งอื่น (redirect)
4xx	Client Error แสดงความผิดพลาดจากการร้องขอของ client
5xx	Server Error แสดงความผิดพลาดจากการทำงานของ Server

สำหรับ client ใน WWW ทั่วไปใช้โปรแกรมที่เรียกว่า Web Browser หรือ Browser ซึ่งทำหน้าที่หลักในการแสดงเอกสาร อาจรับ input จากผู้ใช้ หรือให้ link เชื่อมต่อไปยังเอกสารอื่นก็ได้ ตัวอย่างของ Browser ที่นิยมใช้เช่น Mosaic, Netscape และ Internet Explorer เป็นต้น

Unix/Linux Network Programming

พื้นฐานของการโปรแกรมเครือข่าย [21], [22] คือ process ซึ่งทั่วไป process หมายถึงโปรแกรมที่ถูก executed โดยระบบปฏิบัติการ เมื่อเรากล่าวว่าคอมพิวเตอร์ 2 ตัวกำลังติดต่อสื่อสารกันนั้นแท้จริงแล้วเป็นการติดต่อสื่อสารกันระหว่าง process ซึ่งอาศัยอยู่ในแต่ละเครื่องนั่นเอง

ในระบบปฏิบัติการ Process จะถูกอ้างถึงได้โดยใช้หมายเลข หรือ Process ID โดยที่ระบบปฏิบัติการจะเก็บข้อมูล อาทิ address space, stack space, virtual address space, สำเนาของ register set (เช่น Program Counter (PC), Stack Pointer (SP), Instruction Register (IR), Program Status Word (PSW)

และ processor registers อื่นทั่วไป), ข้อมูลผู้ใช้, สำเนาของโครงสร้างข้อมูลที่เกี่ยวข้องและสถานะปัจจุบันของ process (เช่น waiting, ready, ฯลฯ) เพื่อเก็บสถานะต่าง ๆ ที่เกี่ยวข้องในการทำงานสำหรับแต่ละ process โดยจะถูกเก็บไว้ในส่วน Process Control Block (PCB)

สำหรับการโปรแกรมเครือข่ายจะเกี่ยวข้องกับตั้งแต่ 2 Processes ขึ้นไป โดยที่ Process เหล่านี้จะถูกสร้าง executed และหยุดทำงาน ซึ่งเป็นพื้นฐานที่สำคัญในการทำงานของระบบยูนิกซ์หรือลินุกซ์ ในการโปรแกรมเราสามารถสร้างและทำงานกับ Process ได้โดยใช้คำสั่งระบบที่เกี่ยวข้อง อาทิ

1 คำสั่งระบบ fork

ในการสร้าง Process ใหม่ทำได้โดยการที่ Process ที่กำลังทำงานอยู่เรียกคำสั่ง fork ซึ่งเป็นคำสั่งของระบบ หลังถูกเรียกใช้ fork จะสร้างสำเนาของ Process ซึ่งเรียกคำสั่งนั้นหรือ Parent Process ดังนั้นเราจึงมักเรียก Process ที่สร้างจากการ fork ว่า Child Process สังเกตว่าการเรียกคำสั่ง fork ถูกเรียกเพียงครั้งเดียวใน Parent Process แต่จะตอบกลับ 2 ครั้งคือครั้งหนึ่งใน Parent Process และอีกครั้งใน Child Process ค่าที่คืนกลับจากคำสั่งสำหรับ Parent Process จะเป็นค่า process ID ของ child process ในขณะที่ใน child process คืนค่ากลับเป็น 0 ในกรณีหากการสร้างล้มเหลว fork จะคืนค่ากลับเป็น -1 แทน ในกรณีที่ child process ต้องการทราบ Process ID ของ Parent สามารถถามได้โดยใช้คำสั่ง getppid

เมื่อ Child Process ถูกสร้างจะมีการใช้ข้อมูลบางส่วนร่วมกับ (share) Parent Process ของมัน อาทิ ไฟล์หรือ devices ที่ parent เปิดไว้ก่อนสร้าง child เพื่อให้ง่ายในการส่งต่อไฟล์ไปยัง child

สำหรับการสำเนา Child Process จะสำเนาข้อมูลต่อไปนี้มาจาก Parent Process

- real user ID
- real group ID
- effective user ID
- effective group ID
- process group ID
- terminal group ID
- root directory
- current working directory
- signal handling settings
- file mode creation mask

อย่างไรก็ตาม Child Process มีค่าข้อมูลต่อไปนี้แตกต่างไปจาก Parent Process

- หมายเลข child process ID ใหม่ซึ่งไม่ซ้ำกับที่ระบบมีอยู่เดิม
- หมายเลข parent process ID ที่แตกต่างไปจาก parent process ของมัน
- สำเนาของ parent’s file descriptions

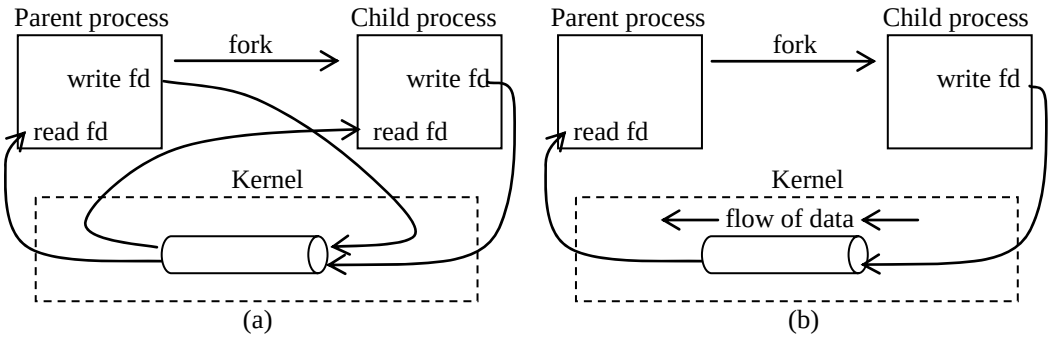
- time left until an alarm clock signal ซึ่งถูกกำหนดใหม่ให้เป็น 0 ใน Child Process

2 คำสั่งระบบ exit

Process อาจยุติการทำงานได้โดยการเรียกคำสั่ง exit ซึ่งจะไม่มีการคืนค่ากลับไปยังผู้เรียก เมื่อ process เรียกคำสั่ง exit ค่าสถานะซึ่งเป็นเลขจำนวนเต็มจะถูกส่งคืนให้กับ kernel นอกจากนี้ค่าสถานะนี้ยังส่งกลับไปยัง parent process ได้อีกด้วยผ่านคำสั่ง wait

3 คำสั่งระบบ pipe

Pipe สร้างทางไหลของข้อมูลแบบทางเดียว เมื่อทำงานแล้วจะคืนค่า 2 file descriptors อยู่ใน filedes[0] และ filedes [1] เปิดไว้เพื่อการเขียน ตัวอย่างการใช้โดยทั่วไป เช่น pipe ถูกสร้างขึ้นเพื่อเป็นทางติดต่อระหว่าง 2 processes โดยที่ parent process สร้าง pipe และ fork เพื่อสร้างสำเนาของ process ของตัวมันเอง จากนั้น parent process ทำการปิดช่องปลายทางการเขียนของ pipe ในขณะที่ child process ปิดปลายทางที่เป็นช่องการอ่านของ pipe ซึ่งทำให้เกิดช่องทางส่งเพียงทางเดียวระหว่าง 2 Processes จาก Child ไปยัง Parent (ดังรูปที่ 2.5)



รูปที่ 2.5 การ fork และ pipe โดยที่ (a) ก่อนการปิด read channel และ write channel (b) หลังการปิดทำให้ได้ช่องการไหลของข้อมูลแบบทางเดียวจาก child process ไปยัง parent process

4 คำสั่งระบบ wait

รูปแบบคำสั่ง `int wait(int *status);`

Process สามารถถูกสั่งให้รอจน Child process ทำงานเสร็จได้โดยใช้คำสั่งระบบ wait ค่าที่คืนกลับจากคำสั่งนี้คือค่า Process ID ของ child process ซึ่งยุติการทำงาน ในกรณีที่ process ไม่มี child process คำสั่ง wait จะคืนค่า -1 ในทันที แต่ในกรณีที่มีตั้งแต่ 1 child process และยังไม่มี child process ใดยุติการทำงาน Parent process นั้นก็จะถูก suspended โดย kernel จนกว่าจะมี child process ใด process หนึ่งยุติการทำงาน ในการยุติการทำงานค่าคืนกลับจากคำสั่ง exit ของ child process จะถูกส่งกลับมาผ่านพารามิเตอร์ status

5 คำสั่งระบบ select

รูปแบบคำสั่ง
```
int select(int maxfdpl, fd_set *readfds,
                    fd_set *writefds, fd_set *exceptfds,
                    struct timeval *timeout);
```

คำสั่ง select ช่วยให้ process ผู้ใช้ แจ้งบอกให้ kernel เรียกปลุก process จากหนึ่งในหลายเหตุการณ์ที่จะเกิดขึ้นซึ่ง process ผู้ใช้สนใจ ตัวอย่างเช่น process ผู้ใช้อาจแจ้งให้ kernel ปลุกเมื่อมีข้อมูลและพร้อมสำหรับการอ่านซึ่งอาจเข้ามาจาก 2 sockets ที่ process ผู้ใช้สนใจ

6 Daemon Process

Daemon Process คือ process ที่ execute อยู่ใน background (กล่าวคือไม่เชื่อมต่อกับ terminal หรือการ login) โดย daemon นี้จะเฝ้ารอเหตุการณ์อย่างใดอย่างหนึ่ง หรืออาจทำงาน background บางอย่างเป็นระยะ ๆ ตัวอย่างเช่น httpd เป็น daemon process ที่เฝ้ารอรับการติดต่อจาก client เพื่อร้องขอบริการโดยใช้ HTTP protocol

Daemon Process อาจถูกเริ่มต้นได้ด้วยหลายวิธีการดังนี้

1. ในช่วงเวลาที่ start up ระบบ Daemon ของระบบส่วนใหญ่เกิดขึ้นโดยการกำหนดไว้ในไฟล์ script /etc/rc ซึ่งจะถูก execute โดย /etc/init เมื่อระบบเริ่มต้น Daemon ที่เกิดในกลุ่มนี้จะได้สิทธิของ superuser
2. จากไฟล์ /user/lib/crontab Daemon ที่ถูกสร้างโดยวิธีนี้สามารถได้สิทธิของ superuser เพราะโปรแกรม cron ทำงานด้วยสิทธิ superuser
3. จากไฟล์ crontab ของผู้ใช้ ซึ่งระบบอาจอนุญาตให้ผู้ใช้ทั่วไปสร้างไฟล์ crontab ของตนเองได้
4. โดยผ่านคำสั่ง at เพื่อกำหนดตารางให้โปรแกรมถูกทำงาน ณ เวลาที่กำหนด
5. จาก terminal ผู้ใช้ ซึ่งอาจเป็น foreground หรือ background job ก็ได้ โดยทั่วไปมักใช้เพื่อการทดสอบระบบ

ลักษณะของ Daemon ทั่วไปมีดังนี้

- ถูกสั่งให้เริ่มต้นเมื่อตอนเริ่มต้นระบบ
- อายุ (lifetime) ของ Daemon มักจะอยู่ตลอดช่วงที่ระบบยังทำงานอยู่ โดยทั่วไปจะไม่ยุติแล้วมาเริ่มต้นใหม่กลางคัน
- ส่วนใหญ่มักใช้เวลาไปกับการรอคอยจนกว่าจะเกิดเหตุการณ์ที่เฝ้ารอฟังอยู่ เพื่อจะได้ทำงานตามบริการที่กำหนดไว้
- ทั่วไปมักสร้าง Process ลูกใหม่ให้เป็นผู้ให้บริการตามคำร้องขอ แทนทำงานด้วยตนเอง

7 การโปรแกรมบน Socket

ในแง่ของการทำงานโดยทั่วไปของ Client/Server บทบาทของ Client และ Server Process จะเป็นแบบ Asymmetric กล่าวคือแต่ละส่วนจะทำงานแตกต่างกันไป โดย Server จะต้องเป็นเริ่มทำงานก่อน และมักจะทำงานเป็นขั้นตอนดังนี้

1. เปิดช่องทางในการสื่อสาร และบอกให้ local host ทราบว่าพร้อมจะรับการติดต่อจาก client บน address ที่ระบุ
2. รอจนกว่าจะมีการร้องขอบริการจาก client ตาม address ที่ระบุ
3. สำหรับ ในกรณีที่ server เลือกที่จะให้บริการนั้นเอง ซึ่งส่วนใหญ่มักเป็นบริการที่สามารถตอบให้แล้วเสร็จได้ภายในการตอบครั้งเดียวและใช้เวลาในการประมวลผลบริการไม่นานนัก
4. สำหรับในกรณีที่การบริการนั้นอาจต้องใช้เวลานานและอาจต้องคงการติดต่อไว้ ในกรณีนี้ server มักเลือกที่จะสร้าง Process ใหม่หรือ child process ขึ้นเพื่อช่วยในการประมวลผลบริการ ตอบผลลัพธ์กลับไปยัง client และเมื่อ process ใหม่ที่สร้างขึ้นให้บริการเรียบร้อยก็จะปิดช่องทางการสื่อสารกับ client ที่ให้บริการพร้อมกับสิ้นสุดการทำงานของ child process ในขณะที่ server process กลับไปรอรับบริการจาก client อื่นต่อไป
5. กลับไปรับการร้องขอบริการในขั้นตอนที่ 2 ใหม่อีกครั้ง

สำหรับ Client process จะทำงานดังนี้

1. เปิดช่องทางการสื่อสารเพื่อติดต่อไปยัง address ที่ระบุ บน host เฉพาะที่ให้บริการ
2. ส่งข้อความการร้องขอไปยัง server และรับผลลัพธ์การทำงานกลับ ทำเช่นนี้ไปเรื่อย ๆ ตามที่ต้องการ
3. ปิดช่องทางการสื่อสารและสิ้นสุดการทำงาน

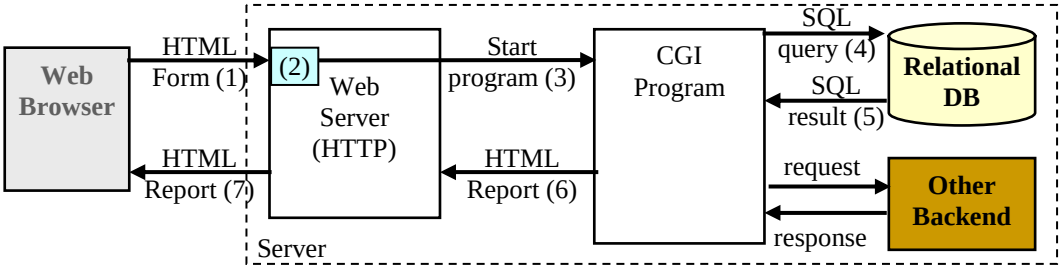
ดังนั้นสำหรับพื้นฐานในการโปรแกรมบน Socket สำหรับ Server อาจสรุปเป็นขั้นตอนดังนี้

1. สร้าง socket โดยอาศัยคำสั่งระบบ
2. bind address ของ server เข้ากับ socket
3. wait หรือรอการเชื่อมต่อจาก client จากนั้น fork child process เพื่อให้บริการที่ร้องขอในขณะที่ parent process กลับไปรอการเชื่อมต่อจาก client อื่น

CGI

เพื่อให้การทำงานของ WWW รองรับการร้องขอเอกสารที่อาจถูกสร้างหรือให้คำตอบเป็นแบบไดนามิค หนึ่งในสถาปัตยกรรมที่นำมาใช้คือ Common Gateway Interface หรือ CGI [23] CGI กำหนดแนวทางมาตรฐานสำหรับ Web Server ในการ execute โปรแกรมบริการโดยอาจรับค่าข้อมูลเข้ามาจากผู้ใช้ องค์ประกอบและวิธีการทำงานของ CGI เพื่อดึงข้อมูลจากระบบจัดการฐานข้อมูล แสดงได้ดังรูปที่

2.6 โดยทั่วไปข้อมูลเข้าจากผู้ใช้ถูกส่งผ่านมาจาก HTML Form พร้อมโปรแกรมดำเนินการบริการนั้นบน Web Server (ขั้นตอนหมายเลข (1)) เมื่อส่งฟอร์มร้องขอมายัง Server Web Server จะประมวลผลคำขอ (ขั้นตอนหมายเลข (2)) โดยแยกพารามิเตอร์ และเริ่ม execute โปรแกรมที่ระบุให้นั้นพร้อมให้ค่าพารามิเตอร์ที่จำเป็นกับโปรแกรม CGI (ขั้นตอนหมายเลข (3)) โปรแกรม CGI เมื่อได้รับคำร้องขอดำเนินการประมวลผลตามบริการของตนเอง ซึ่งอาจต้องทำคิวรีไปยังระบบจัดการฐานข้อมูล (ขั้นตอนหมายเลข (4)) เพื่อนำข้อมูลที่ได้รับตอบกลับมาจากระบบจัดการฐานข้อมูล (ขั้นตอนหมายเลข (5)) มาประมวลผลและจัดรูปแบบใหม่สร้างเป็นรายงาน HTML ส่งกลับไปยัง Web Server (ขั้นตอนหมายเลข (6)) เพื่อส่งต่อไปยัง client (ขั้นตอนหมายเลข (7)) ที่ร้องขอบริการนั้นเพื่อนำเสนอผลลัพธ์



(1) Get document request sent to the server
(2) Process input
(3) Start CGI program to fetch documents
(4) SQL query to the database
(5) SQL result from the database
(6) Generate HTML report as a response to the server
(7) Forward the HTML document to the client

รูปที่ 2.6 องค์ประกอบและการทำงานของ CGI

หลักการเขียนโปรแกรม CGI โดยทั่วไปการสร้างโปรแกรม CGI ทำโดยสร้างไดเร็กทอรี CGI บน Server ซึ่งส่วนใหญ่ดำเนินการเมื่อตอน setup การระบุไดเร็กทอรีนี้เพื่อแจ้งให้ server ทราบว่าไฟล์ที่เป็นไดเร็กทอรีบนสุดของโปรแกรม CGI อยู่ที่ใด เมื่อมีการร้องของจากผู้ใช้จะได้ไปยังที่ตั้งที่ถูกต้อง (โดยทั่วไป default directory คือ /cgi-bin/) ดังนั้นเมื่อมีการร้องขอ URLs เช่น http://www.cs.kmitnb.ac.th/cgi-bin/search จะช่วยให้ไปยังโปรแกรม CGI ได้ถูกต้อง อย่างไรก็ตามไดเร็กทอรีนี้สามารถตั้งเป็นชื่อใด ๆ ก็ได้ ในบาง Server สามารถที่จะ set up โดยไม่ต้องกำหนดไดเร็กทอรี แต่ใช้นามสกุลของไฟล์ .cgi เป็นตัวกำหนดแทนได้

โปรแกรม CGI เป็นเหมือน executable program ทั่วไป ดังนั้นจึงสามารถถูกโปรแกรมได้โดยใช้ภาษาการโปรแกรม อาทิภาษา C, C++, java, Perl, ฯลฯ ได้ จากที่กล่าวไว้ในหลักการทำงานข้างต้นก่อนที่โปรแกรม CGI จะถูกเริ่มต้น executed Web server จะ set ตัวแปรสิ่งแวดล้อม (environment variables) ซึ่งให้ข้อมูลเกี่ยวกับการร้องขออาทิ IP address ของ client, ข้อมูลบน header ของการร้องขอนั้น และถ้า URL requested มีเครื่องหมาย ? ค่าของข้อมูลที่อยู่ตามหลัง ? ก็จะถูกส่งไปเป็น environment variable ด้วย

Environment Variable อาจแบ่งออกได้เป็น 3 ประเภท

- ตัวแปรข้อมูลของ Server: เป็นข้อมูลเกี่ยวกับ server โดยตัวแปรเหล่านี้ได้แก่
  - *SERVER_SOFTWARE* ให้ชื่อและเวอร์ชันของ server software เช่น Apache/1.3.33
  - *SERVER_NAME* ซึ่งให้ชื่อของ Server โดยทั่วไปเป็นชื่อ hostname หรือ IP Address เช่น www.cs.kmitnb.ac.th หรือ 202.44.40.200
  - *GATEWAY_INTERFACE* แสดงเวอร์ชันของ CGI ที่ server สนับสนุน เช่น CGI/1.1
  - *SERVER_PROTOCOL* ให้ชื่อและเวอร์ชันของโปรโตคอลของ server เช่น HTTP/1.0
  - *SERVER_PORT* ให้หมายเลขพอร์ตที่เชื่อมต่อมายัง Server ทั่วไปคือพอร์ต 80
- ตัวแปรข้อมูลของ Client: เป็นข้อมูลเกี่ยวกับ client โดยตัวแปรเหล่านี้ได้แก่
  - *REMOTE_HOST* ซึ่งให้ชื่อเต็มของโดเมนของ client ที่ร้องขอบริการ ในกรณีที่ไม่มีชื่อค่าของตัวแปรจะถูกกำหนดให้เป็น null
  - *REMOTE_IP* ให้ค่า IP address ของ client ที่ร้องขอบริการ
  - *REMOTE_USER* ให้ชื่อของผู้ใช้ในกรณีสำหรับร้องขอเอกสารที่มีการป้องกัน
  - *REMOTE_IDENT* ให้ค่าชื่อผู้ใช้บน server ในกรณีที่ server สนับสนุน RFC 931 identification ตัวแปรนี้จึงจะถูกกำหนดขึ้น โดยการใช้ควรจำกัดเพื่อการ log in เท่านั้น
  - *HTTP_USER_AGENT* ให้ชื่อและเวอร์ชันของ Browser ของ client ที่ร้องขอบริการ เช่น Mozilla/2.01
  - *HTTP_ACCEPT* ให้ชื่อชนิด MIME ที่ client รับได้เช่น img/gif, img/jpg โดยค่านี้ได้มาจาก HTTP Header
- ตัวแปรข้อมูลของ CGI script: เป็นตัวแปรข้อมูลเกี่ยวกับฟอร์มและ script ได้แก่
  - *REQUEST_METHOD* ให้ชื่อ HTTP method ที่ถูกร้องขอ (จากฟอร์ม) เช่น GET, HEAD หรือ POST เป็นต้น
  - *PATH_INFO* ให้ข้อมูลเพิ่มเติมเกี่ยวกับ path ซึ่งส่งมาจาก client หรือกล่าวอีกนัยหนึ่งคือ scripts ถูกเข้าถึงโดยผ่าน virtual pathname ตามด้วยชื่อเพิ่มเติมของ path ค่าชื่อเพิ่มเติมนี้จะถูกส่งผ่านมากับพารามิเตอร์ PATH_INFO ตัวอย่างเช่นถ้า CGI script: test-cgi ถูกร้องขอผ่าน http://www.cs.kmitnb.ac.th/cgi-bin/test-cgi/docs/index.html ค่าของ PATH_INFO ในกรณีนี้คือ /docs/index.html
  - *PATH_TRANSLATED* ให้ค่าการแปลง PATH_INFO พร้อม virtual pathname ให้เป็น physical pathname ของเอกสารนั้น ดังนั้นถ้า root ของเอกสารบน server อยู่ที่ /usr/local/etc/httpd/htdocs สำหรับค่า PATH_INFO = docs/index.html จะได้ PATH_TRANSLATED เป็น /usr/local/etc/httpd/htdocs/cgi-bin/docs/index.html

- *SCRIPT_NAME* ให้ค่า virtual path ของ script ที่ถูกสั่งให้ทำงาน ใช้เพื่อการอ้างอิงถึงURL ของตัวเอง
- *QUERY_STRING* ให้ข้อมูลเกี่ยวกับ query ซึ่งเป็นค่าที่ตามหลังเครื่องหมาย ? ใน URL โดยถูกกำหนดไว้ในรูป field-value pair ในกรณีที่มีมากกว่า 1 ตัวแต่ละคู่จะคั่นกันด้วยเครื่องหมาย & เช่น `cgi-bin/plus.cgi?m=14&n=15` ค่าของ query string คือ m=14 และ n=15
- *CONTENT_TYPE* ให้ค่าของชนิด MIME ที่ส่งมากับการร้องขอ
- *CONTENT_LENGTH* ให้ข้อมูลเกี่ยวกับความยาวของข้อมูลการร้องขอ ในกรณีที่มีข้อมูลที่ส่งมาด้วย ถ้าไม่มีค่านี้จะเป็น null

ในการคืนคำตอบกลับ โปรแกรม CGI คืนค่าผลลัพธ์เป็น HTTP headers ร่วมกับเอกสาร HTML ไปยัง server โดยการเขียนลงบน standard output ของมัน ตัวอย่างเช่น ในกรณีภาษา Perl หรือ Python script ก็เพียงใช้ประโยค print สำหรับในภาษา C ใช้ printf หรือในภาษา C++ ใช้ cout << เป็นต้น

ตัวอย่างฟอร์มสำหรับเรียกใช้ CGI เพื่อบวกเลข 2 จำนวน

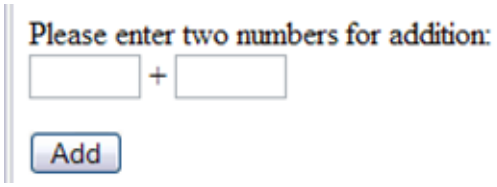
```
<HTML><TITLE>Sample CGI for addition</TITLE>
<BODY>
<FORM ACTION="http://www.cs.kmitnb.ac.th/cgi-bin/plus.cgi">
<P>Please enter two numbers for addition: <BR>
<INPUT NAME="m" SIZE="5"> +
<INPUT NAME="n" SIZE="5">
<P>
<INPUT TYPE="SUBMIT" VALUE="Add">
</FORM>
</BODY>
</HTML>
```

ตัวอย่างโปรแกรม CGI เพื่อบวกเลข 2 จำนวนในภาษาซี

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
      char *data;
      long m, n;
      printf("%s%c%c\n",
      "Content-Type:text/html;charset=iso-8859-1",13,10);
      printf("<TITLE>Addition result</TITLE>\n");
      printf("<H1>Addition result</H1>\n");
      data = getenv("QUERY_STRING");
      if(data == NULL)
         printf("<P>Error! in parsing input data");
      else if(sscanf(data,"m=%ld&n=%ld",&m,&n)!=2)
         printf("<P>Error! Invalid input. Allow numeric only.");
      else
         printf("<P>The addition of %ld and %ld is %ld.",m,n,m+n);
      return 0;
}
```

ซึ่งเมื่อมีการเรียกใช้ผ่าน web browser จะมีหน้าจอดังรูปที่ 2.7 และเมื่อผู้ใช้กรอกค่าตัวเลข 2 จำนวน เช่น 14 และ 15 ทำการ submit ผ่านปุ่ม Add จะทำให้มีการร้องขอไปยัง server และเมื่อประมวลผลเสร็จจะคืนค่ากลับและให้ผลลัพธ์ดังหน้าจอในรูปที่ 2.8



รูปที่ 2.7 แสดงฟอร์มของโปรแกรม CGI สำหรับบวกเลข 2 จำนวน

รูปที่ 2.8 แสดงผลลัพธ์การทำงานของโปรแกรม CGI สำหรับบวกเลข 2 จำนวน

การเข้ารหัสภาษาไทยสำหรับ URL

ทั่วไปสำหรับข้อความที่ส่งผ่านอินเทอร์เน็ต ในภาษาของชาติที่แตกต่างกันมีการเข้ารหัสที่แตกต่างกันไป ดังนั้นในกรณีที่ข้อความมีภาษาอื่นเช่น ภาษาไทยอยู่ในข้อความ จำเป็นจะต้องทำการเข้ารหัสข้อความเหล่านั้นเสียก่อนเพื่อให้ Server เข้าใจข้อความได้ถูกต้องตรงกัน ดังนั้นสำหรับข้อความภาษาไทยหากนำไปใช้เป็น URL สำหรับการค้นหา จึงต้องเข้ารหัสให้ถูกต้องตามมาตรฐาน RFC 3986 ซึ่งจะแปลงข้อความให้อยู่ในรูป percent-encoding (หรือ %XX) เพื่อให้สามารถใช้เป็นคิวรีบน Search Engine ทั่วไปได้

หลักการเข้ารหัสตามมาตรฐาน RFC 3986 [18] ได้จัดแบ่งกลุ่มตัวอักษรออกเป็น Reserved และ Unreserved เท่านั้นที่สามารถใช้เป็น URL ได้โดยตรง (ตารางที่ 2.3) ตัวอักษร ASCII ที่ไม่ใช่ Reserved หรือ Unreserved Characters จะต้องถูกเปลี่ยนให้อยู่ในรูปของ UTF-8 โดยใช้หลักการแปลงรหัส Unicode และ UTF-8 รวมถึงจำนวนไบท์ที่ต้องใช้ตาม

ตารางที่ **2.4**

ตารางที่ 2.3 แสดงกลุ่มตัวอักษรประเภท Reserved และ Unreserved ที่ไม่ต้องเข้ารหัส

ประเภท	ตัวอักษร
Reserved Characters	: / ? # [ ] @ ! $ & ' ( ) * + , ; =
Unreserved Characters	อักษร US-ASCII (หรือ A-Z, a-z, 0-9) - _ . ~

ตารางที่ 2.4 แสดง รหัส Unicode และ UTF-8

ค่า Unicode ของ UTF-8	จำนวนไบท์ของ UTF-8 ที่ต้องใช้	รูปแบบไบนารี (Binary Representation)
0 - 127	1 byte	0XXX.XXXX
128 - 2047	2 bytes	110X.XXXX 10XX.XXXX
2048 - 65535	3 bytes	1110.XXXX 10XX.XXXX 10XX.XXXX
65536 - 2097151	4 bytes	1111.0XXX 10XX.XXXX 10XX.XXXX 10XX.XXXX

จากรูปแบบไบนารีที่ได้ ต้องแปลงให้อยู่รูปเลขฐานสิบหกในรูปแบบ **%**XX โดย XX เป็นค่าเลขฐานสิบหกของแต่ละไบท์

การทำ caching

การทำ caching เป็นการเก็บคำตอบไว้เผื่อการเรียกถามซ้ำในอนาคต ทำให้ไม่จำเป็นต้องดึงข้อมูลเหล่านั้นมาจากแหล่งต้นตอใหม่ทุกครั้ง ปัจจัยสำคัญที่ต้องพิจารณาในการทำ caching คือ *ความถูกต้อง (Consistency)* ของคำตอบ เนื่องจากคำตอบนั้นอาจถูกเก็บไว้ในที่ตั้งชั่วคราว ในขณะที่ข้อมูลที่แหล่งต้นตอได้ถูกปรับปรุงหรือเปลี่ยนแปลงไปแล้ว เป็นผลให้คำตอบที่ได้รับขาดความถูกต้องกับค่าปัจจุบัน สำหรับการทำ caching บน server ตัวกลางซึ่งไม่ใช่แหล่งต้นตอข้อมูล อาจเลือกตรวจสอบความถูกต้องโดยการส่งคิวรีไปยังแหล่งต้นตอ เพื่อตรวจสอบการเปลี่ยนแปลงอีกครั้งก่อน หากไม่เปลี่ยนแปลงก็เลือกส่งคำตอบที่ cache ไว้ส่งไปยังผู้ใช้ได้เลย ในกรณีนี้จะเห็นว่าประสิทธิภาพการทำงานจะไม่ดีนัก เนื่องจากต้องส่ง message ไปถามแหล่งต้นตอก่อน และหากข้อมูลถูกเปลี่ยนแปลงไปแล้ว ก็จำเป็นจะต้องส่งคิวรีเพื่อหาคำตอบใหม่จากแหล่งต้นตออีกครั้ง แนวทางอีกอย่างหนึ่งที่ทำได้คือการกำหนดช่วงอายุของ cache ในกรณีนี้การบำรุงรักษา consistency จัดว่าเป็นแบบ weak consistency ในขณะที่แบบแรกเป็นแบบ strong consistency ในกรณีของ weak consistency จะต้องยอมรับว่าข้อมูลอาจขาดความทันสมัยและถูกต้องในช่วงเวลาหนึ่ง ตัวอย่างการทำ caching แบบนี้ใช้ในกรณี web proxy caching [23] โดยที่ server จะกำหนดช่วงเวลาการหมดอายุดังสมการ (3)

$$T_{\text{expire}} = \alpha(T_{\text{cached}} - T_{\text{last\_modified}}) + T_{\text{cached}} \qquad (3)$$

โดย $T_{\text{last\_modified}}$ คือเวลาล่าสุดที่เอกสารถูกปรับปรุง

T_{cached} คือเวลาล่าสุดที่เอกสารนั้นถูกบันทึกลงใน cache

T_{expire} คือเวลาที่หมดอายุ

α คือค่าน้ำหนักที่ใช้ถ่วงสำหรับกำหนดความสำคัญของผลต่างของเวลาที่เก็บบันทึกใน cache กับเวลาล่าสุดของเอกสาร ซึ่งในทางปฏิบัติพบว่าควรเป็น 0.2

ตราบเท่าที่ข้อมูลใน cache ยังไม่หมดเวลา เราจะใช้คำตอบจาก cache แทนการร้องขอใหม่จาก server แต่หลังจากหมดเวลาแล้ว จะต้องร้องขอเอกสารใหม่จาก server แทนค่าใน cache

อย่างไรก็ตาม การ cache ข้อมูลไว้จะเป็นประโยชน์ในกรณีที่มีการร้องขอซ้ำอาจเป็นจากผู้ใช้เดิมหรือผู้ใช้อื่น แต่จะไม่เป็นประโยชน์ในกรณีที่การร้องขอซ้ำมีน้อยหรือไม่เกิดขึ้นเลย ดังนั้นการร้องขอแล้วพบว่าข้อมูลอยู่ใน cache หรือที่เรียกว่า cache hit หากอัตราการ hit มีค่าต่ำ ข้อมูลใน cache มีประโยชน์น้อย เพราะไม่พบใน cache ทำให้ต้องร้องขอจากแหล่งต้นตอบ่อย ๆ การไม่พบใน cache นี้อาจถูกเรียกว่า cache miss ปัจจัยที่สำคัญอีกอย่างหนึ่งของการทำ caching คือการพยายามเก็บข้อมูลซึ่งจะถูกร้องขอบ่อย ๆ ไว้เพื่อ maintain ให้ cache hit มีค่าสูง

ปัจจัยในเรื่องของ cache อีกอย่างคือการตรวจสอบว่าสิ่งที่ร้องขอ จับคู่ได้กับสิ่งที่เก็บไว้ใน cache หรือไม่ เทคนิคหนึ่งที่ช่วยให้การตรวจสอบเป็นไปอย่างมีประสิทธิภาพเช่น*การใช้เทคนิค Hashing* โดยนำค่าที่ใช้ในการตรวจสอบไปผ่าน hash function เพื่อหาค่า hash ที่จะใช้อ้างถึงกับสิ่งที่เก็บ ในกรณีที่สิ่งสืบค้นมีค่า hash ไม่ตรงกันกับที่มีอยู่ใน cache แสดงว่าไม่ใช่สิ่งเดียวกัน อย่างไรก็ตามหากค่า hash ตรงกันก็ไม่จำเป็นเสมอไปว่าจะต้องเป็นสิ่งเดียวกัน โดยทั่วไปเทคนิค hashing ช่วยลดปริมาณสิ่งที่ต้องค้นโดยจัดเป็นกลุ่มไว้ตามค่า hash ของมันนั่นเอง

Regular Expression

Regular Expression เป็นเครื่องมือที่ใช้ในการประมวลผลข้อความ (Text) ที่มีประสิทธิภาพและให้ความยืดหยุ่นสูง โดยทั่วไปเครื่องมือนี้ใช้แพทเทิร์นที่ใช้อธิบาย ตรวจสอบหรือตัดข้อความที่ต้องการได้ ในภาษาโปรแกรมยุคใหม่ เช่น จาวา, Perl, PHP มีเครื่องมือที่ช่วยในการทำ Regular Expression Processing โดยสนับสนุนการค้นหา เลือกคำที่ต้องการ หรือเปลี่ยนแปลงข้อความใน Text ข้อมูล

ในการประมวลผล Regular Expression จะใช้แพทเทิร์น ซึ่งกำหนดรูปแบบของข้อความซึ่งจะนำไปใช้เป็นแบบในการเปรียบเทียบเพื่อค้นหา ตัด หรือเปลี่ยนแปลงข้อความ แพทเทิร์นเหล่านี้ถูกกำหนดไว้เป็นแบบแผนสรุปได้ดังตารางที่ 2.5 อย่างไรก็ตามเครื่องมือของ Regular Expression อาจมีความแตกต่างกันไปบ้าง (รายละเอียดและหลักการทำงานของ Regular Expression ดูเพิ่มเติมได้ใน [7])

ตารางที่ 2.5 รูปแบบโดยสรุปของคำคีย์ที่ใช้ในการสร้างแพทเทิร์นสำหรับ regular-expression

คำคีย์	**คำอธิบายถึงลักษณะการจับคู่ (Matches)**
ประเภทตัวอักษร	
x	ตัวอักษร *x*
`\\`	ตัวอักษร backslash (\)
`\0`*n*	ตัวอักษรที่มีค่าเป็นเลขฐานแปด 0*n* (0 <= *n* <= 7)
`\0`*nn*	ตัวอักษรที่มีค่าเป็นเลขฐานแปด 0*nn* (0 <= *n* <= 7)

ตารางที่ 2.5 รูปแบบโดยสรุปของคำคีย์ที่ใช้ในการสร้างแพทเทิร์นสำหรับ regular-expression

คำคีย์	**คำอธิบายถึงลักษณะการจับคู่ (Matches)**
\0*mnn*	ตัวอักษรที่มีค่าเป็นเลขฐานแปด 0*mnn* (0 <= *m* <= 3, 0 <= *n* <= 7)
\x*hh*	ตัวอักษรที่มีค่าเป็นเลขฐานสิบหก 0x*hh*
\u*hhhh*	ตัวอักษรที่มีค่าเป็นเลขฐานสิบหก 0x*hhhh*
\t	ตัวอักษร tab ('\u0009')
\n	ตัวอักษรขึ้นบรรทัดใหม่ (line feed) ('\u000A')
\r	ตัวอักษร carriage-return ('\u000D')
\f	ตัวอักษร form-feed ('\u000C')
\a	ตัวอักษรเสียงกระดิ่ง (bell character) ('\u0007')
\e	ตัวอักษร escape ('\u001B')
\c*x*	ตัวอักษรควบคุมที่มีค่าสอดคล้องกับ *x*
กลุ่ม Character	
[abc]	a, b, หรือor c (กลุ่มอย่างง่าย)
[^abc]	ตัวอักษรใด ๆ ยกเว้น a, b หรือ c (negation)
[a-zA-Z]	กลุ่มแบบ inclusive แสดงตัวอักษรระหว่าง a ถึง z หรือ A ถึง Z
[a-d[m-p]]	กลุ่มแบบ union แสดงตัวอักษรระหว่าง a ถึง d หรือ m ถึง p หรือเท่ากับ [a-dm-p]
[a-z&&[def]]	กลุ่มแบบ intersection แสดงตัวอักษร d, e หรือ f
[a-z&&[^bc]]	กลุ่มแบบsubtraction แสดงตัวอักษร a ถึง z ยกเว้น b และ c หรือเท่ากับ[ad-z]
[a-z&&[^m-p]]	กลุ่มแบบsubtraction แสดงตัวอักษร a ถึง z และไม่รวม m ถึง p หรือเท่ากับ [a-lq-z]
ตัวอักษรที่กำหนดไว้แล้ว (Predefined Character)	
.	ตัวอักษรใด ๆ ซึ่งอาจจับหรือไม่จับคู่กับ line terminators
\d	ตัวเลข : [0-9]
\D	ไม่ใช่ตัวเลข : [^0-9]
\s	ตัวอักษรช่องว่างแบบต่าง ๆ (white space) : [\t\n\x0B\f\r]
\S	ตัวอักษรที่ไม่ใช่ช่องว่าง : [^\s]
\w	ตัวอักษรของคำภาษาอังกฤษ : [a-zA-Z_0-9]
\W	ตัวอักษรที่ไม่ใช่คำภาษาอังกฤษ [^\w]

ตารางที่ 2.5 รูปแบบโดยสรุปของคำคีย์ที่ใช้ในการสร้างแพทเทิร์นสำหรับ regular-expression

คำคีย์	**คำอธิบายถึงลักษณะการจับคู่ (Matches)**
ตัวอักษร POSIX character (US-ASCII เท่านั้น)	
\p{Lower}	ตัวอักษรพิมพ์เล็ก : [a-z]
\p{Upper}	ตัวอักษรพิมพ์ใหญ่ : [A-Z]
\p{ASCII}	ทุก ASCII : [\x00-\x7F]
\p{Alpha}	ตัวอักษรภาษาอังกฤษทุกตัว : [\p{Lower}\p{Upper}]
\p{Digit}	ตัวเลข : [0-9]
\p{Alnum}	ตัวอักษรภาษาอังกฤษและตัวเลข : [\p{Alpha}\p{Digit}]
\p{Punct}	ตัวเชื่อม : !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
\p{Graph}	ตัวอักษรที่เห็นได้ : [\p{Alnum}\p{Punct}]
\p{Print}	ตัวอักษรที่พิมพ์ได้ : [\p{Graph}\x20]
\p{Blank}	ช่องว่างและ tab : [\t]
\p{Cntrl}	ตัวอักษรควบคุม : [\x00-\x1F\x7F]
\p{XDigit}	ตัวเลขฐานสิบหก : [0-9a-fA-F]
\p{Space}	ช่องว่างแบบต่าง ๆ : [\t\n\x0B\f\r]
จับคู่เป็นขอบเขต (Boundary matchers)	
^	เริ่มต้นบรรทัด
$	สิ้นสุดบรรทัด
\b	ขอบเขตเป็นคำ
\B	ขอบเขตไม่เป็นคำ
\A	เริ่มต้นของ input
\G	จุดสิ้นสุดของการจับคู่ก่อนหน้า
\Z	จุดสิ้นสุดของ input แต่สำหรับตัว terminator สุดท้ายหากมี
\z	จุดสิ้นสุดของ input
Greedy quantifiers	
X?	*X*, ไม่ปรากฏหรือปรากฏเพียงหนึ่ง
*X**	*X*, ไม่ปรากฏหรือหลายครั้ง
X+	*X*, อย่างน้อยหนึ่งครั้ง
X{*n*}	*X*, ปรากฏทั้งหมด *n* ครั้ง
X{*n*,}	*X*, ปรากฏอย่างน้อย *n* ครั้ง

ตารางที่ 2.5 รูปแบบโดยสรุปของคำคีย์ที่ใช้ในการสร้างแพทเทิร์นสำหรับ regular-expression

คำคีย์	**คำอธิบายถึงลักษณะการจับคู่ (Matches)**
X{*n*,*m*}	*X*,
Reluctant quantifiers	
X??	*X*, หนึ่งหรือไม่ปรากฏเลย
*X**?	*X*, ไม่ปรากฏหรือหลายครั้ง
X+?	*X*, อย่างน้อยหนึ่งครั้ง
X{*n*}?	*X*, ปรากฏทั้งหมด *n* ครั้ง
X{*n*,}?	*X*, ปรากฏอย่างน้อย *n* ครั้ง
X{*n*,*m*}?	*X*, อย่างน้อย *n* ครั้งแต่ไม่มากกว่า *m* ครั้ง
Possessive quantifiers	
X?+	*X*, หนึ่งหรือไม่ปรากฏเลย
*X**+	*X*, ศูนย์หรือหลายครั้ง
X++	*X*, หนึ่งหรือหลายครั้ง
X{*n*}+	*X*, ปรากฏเท่ากับ *n* ครั้ง
X{*n*,}+	*X*, อย่างน้อย *n* ครั้ง
X{*n*,*m*}+	*X*, อย่างน้อย *n* แต่ไม่มากกว่า *m* ครั้ง
Logical operators	
XY	*X* ตามด้วย *Y*
X|*Y*	*X* หรือ *Y*
(*X*)	X, เป็นกลุ่มที่ต้องการตัดเก็บ (capturing group)
Back references	
n	อะไรก็ตามที่ match กลุ่มที่ n^{th} ที่ต้องการเก็บ (capturing group)
การทำ Quotation	
\	ไม่ทำอะไร แต่ quotes ตัวอักษรที่อยู่ถัดไป
\Q	ไม่ทำอะไร แต่ทำ quotes ทุกตัวอักษรจนถึง \E
\E	ไม่ทำอะไร แต่ปิดท้าย quoting ที่เริ่มต้นด้วย \Q
Special constructs (non-capturing)	
(?:*X*)	*X*, ไม่ใช่กลุ่มที่ต้องการเก็บ (non-capturing group)
(?idmsux-idmsux)	ไม่ทำอะไร แต่เปลี่ยน match flags เป็น on – off
(?idmsux-idmsux:*X*)	*X*, ไม่ใช่กลุ่มที่ต้องการเก็บพร้อมเปลี่ยน flags เป็น on - off

ตารางที่ 2.5 รูปแบบโดยสรุปของคำคีย์ที่ใช้ในการสร้างแพทเทิร์นสำหรับ regular-expression

คำคีย์	**คำอธิบายถึงลักษณะการจับคู่ (Matches)**
(?=*X*)	*X*, ผ่าน zero-width positive lookahead
(?!*X*)	*X*, ผ่าน zero-width negative lookahead
(?<=*X*)	*X*, ผ่าน zero-width positive lookbehind
(?<!*X*)	*X*, ผ่าน zero-width negative lookbehind
(?>*X*)	*X*, ที่เป็นอิสระ, ไม่ใช่กลุ่มที่ต้องการเก็บ

เทคโนโลยีการจัดการระบบฐานข้อมูล

ระบบจัดการฐานข้อมูล (Database Management System (DBMS)) หมายถึง ซอฟต์แวร์ที่ใช้ในการกำหนดสร้าง จัดการและบำรุงรักษาฐานข้อมูลด้วยคอมพิวเตอร์เพื่อให้สามารถใช้ร่วมกันจากหลาย processes ของผู้ใช้อย่างมีประสิทธิภาพ

ระบบฐานข้อมูล มีการโมเดลข้อมูลในระดับแนวคิดในหลายรูปแบบ อาทิ Entity-Relationship Model ซึ่งนิยมใช้ในการออกแบบระบบฐานข้อมูล อย่างไรก็ตามในแง่การทำ Implement โมเดลข้อมูลที่ได้รับความนิยมสูงสุดในการนำไปใช้งานจริงได้แก่ โมเดลแบบรีเลชันนัล ซึ่งเป็นโมเดลที่ระบบจัดการฐานข้อมูลที่ได้รับความนิยม อาทิ Oracle, SQL Server, MySQL หรือระบบจัดการฐานข้อมูลขนาดเล็ก เช่น MS Access ใช้

ระบบจัดการฐานข้อมูลแบบรีเลชันนัลใช้โมเดลในรูปของตาราง ประกอบด้วยแถวหรือรายการ และฟิลด์ หรือ column ในแต่ละรายการแทนสิ่งหรือ entity และแต่ละฟิลด์แสดงรายละเอียดของ entity เหล่านั้น ในแง่การจัดการฐานข้อมูล ระบบจัดการฐานข้อมูลแบบรีเลชันนัลมีภาษาคิวรีเพื่อช่วยให้นักพัฒนาสามารถติดต่อ สร้าง ปรับปรุง หรืออ่านข้อมูลได้ ภาษาที่ได้รับการยอมรับและเป็นมาตรฐานในการติดต่อกับระบบจัดการฐานข้อมูล คือ SQL

SQL เป็นภาษาที่ถูกออกแบบโดยมีวัตถุประสงค์เพื่อให้เข้าใจได้ง่ายและใกล้เคียงกับภาษามนุษย์ ดังนั้นรูปแบบคำสั่งจึงคล้ายกับประโยคภาษาอังกฤษ และเป็นภาษาที่ case-insensitive นั่นคือคำสั่งที่ใช้อักษรตัวพิมพ์ใหญ่หรือตัวพิมพ์เล็กจะให้ความหมายเดียวกัน นอกจากนี้ SQL ถือได้ว่าเป็น non-procedural language หรือภาษาอธิบายข้อกำหนดความต้องการโดยไม่ต้องอธิบายขั้นตอนการทำงาน

ตัวอย่างของภาษา SQL เพื่อใช้ในการสร้างตาราง keyword เพื่อใช้ในการเก็บข้อมูลคำค้น ประกอบด้วย รหัสซึ่งไม่ซ้ำ และเป็นคีย์หลักของรายการ คำค้น flag found เพื่อระบุว่าเป็นคำค้นที่มีคำตอบหรือไม่ วันที่เข้าถึงครั้งสุดท้าย และวันที่สร้าง

```
CREATE TABLE keyword (
  id     VARCHAR(50) NOT NULL,
  keyword   VARCHAR(255),
  found     CHAR,
  accessDate   DATE,
```

```
  createDate   DATE,
  PRIMARY KEY (id),
)
```

ในการค้นหาทำได้โดยใช้คำสั่ง SQL SELECT ซึ่งมีรูปแบบคำสั่งอย่างง่ายดังนี้

```
SELECT  A1, A2,…,An
FROM    r1, r2, …, rm
WHERE   conditions P
```

โดยที่ A1, A2, …, An คือ attribute list

r1, r2, …, rm คือ list ของตาราง

conditions P คือเงื่อนไขในการเลือกรายการ

เมื่อทำคิวรีนี้เสร็จแล้ว ผลลัพธ์ที่ได้เป็นตารางใหม่ที่มี attributes คือ A1, A2,... ,An

รายละเอียดเกี่ยวกับระบบจัดการฐานข้อมูลสามารถอ่านเพิ่มเติมได้จากหนังสือระบบจัดการฐานข้อมูลทั่วไป [20]

งานวิจัยที่เกี่ยวข้อง

มีผลงานการวิจัยตลอดจนการค้นหาโดย MetaSearch ในต่างประเทศเป็นจำนวนมาก หากแต่ Metasearch เหล่านั้นค้นหาเฉพาะจากแหล่งที่ใช้ภาษาอังกฤษเท่านั้น วิธีการรวบรวมข้อมูลหลายแหล่งที่ใช้ก็มีแตกต่างกันไป ในบาง Metasearch จะส่งการค้นหาโดยไม่รวบรวมสิ่งที่ค้นหาทำให้ได้ข้อมูลที่อาจมีความซ้ำซ้อน ตัวอย่างของ Metasearch เหล่านี้ได้แก่ 1Blink [1], All4One[2], Metasearch[[17] จะค้นหาจากแหล่งข้อมูลโดยไม่ Integrated ข้อมูล นอกจากนี้บาง Metasearch จะค้นหาและรวบรวมแต่ก็ไม่มีการสนับสนุนภาษาไทย ตัวอย่างเช่น SavvySearch[[19], Argus Music Searcher [5], Highway61 [10], MetaCrawler [16] เป็นต้น

บทที่ 3

วิธีการวิจัย (Methodology)

ในการสร้าง Multiple Search Engine มีขั้นตอนการดำเนินการดังนี้

ศึกษาและเลือก General Purposed Search Engine

เพื่อสร้าง Multiple Search Engine ซึ่งต้องอาศัยความสามารถในการค้นคำของ General Purposed Search Engine จึงจำเป็นต้องพิจารณาเลือกหา Search Engine ที่เหมาะสมเพื่อให้เป็นแหล่งสืบค้น โดยได้พิจารณาลักษณะของ Search Engine พบว่า AltaVista [3] HotBot [11] และ Lycos [14] ซึ่งเป็น Search Engine ขนาดใหญ่และได้รับความนิยม เราจึงเลือก Search Engine ทั้ง 3 เป็นหลัก

ในแต่ละ Search Engine มีรูปแบบการส่งคิวรีเพื่อการค้นหาและการตอบกลับการค้นคำในรูปแบบที่แตกต่างกัน ดังนั้นจึงจำเป็นต้องศึกษารูปแบบการทำคิวรีคำค้นสรุปได้เป็นตารางที่ 3.1โดย %s คือข้อความของคำค้น

ตารางที่ 3.1 แพทเทิร์น String สำหรับการคิวรี

Search Engine	**คิวรี String**
AltaVista	`http://www.altavista.com/cgi-bin/query?pg=q&kl=XX&q=%s HTTP/1.0\n`
HotBot	`http://www.hotbot.com/default.asp?MT=%s HTTP/1.0\n`
Lycos	`http://www.lycos.com/cgi-bin/pursuit?matchmode=and&cat=lycos&query=%s HTTP/1.0\n`

และรูปแบบการตอบกลับซึ่งต้องหาแพทเทิร์น string สำหรับการตัดคำตอบสรุปได้ดังตารางที่ 3.2 สำหรับรายละเอียดในการเรื่องคำค้นและการตัดคำจะได้อธิบายในส่วนต่อไป

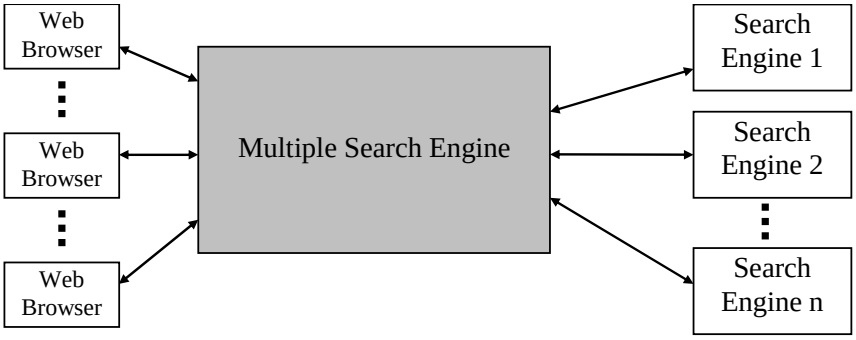
ตารางที่ 3.2 แพทเทิร์นการตัด String เพื่อหาจุดเริ่มต้นและหา Link ของคำตอบ

Search Engine	**จุดเริ่มแรกของ URL คำตอบ**	**ข้อมูลคำตอบ**
AltaVista	String เริ่มต้นจะอยู่หลังจาก tag “<dl><dt>”	อยู่ระหว่าง tag <a> … โดยจะต้องตัดส่วนของ HTML tag ทิ้งไป
Hotbot	String เริ่มต้นจะอยู่หลังจาก “director.asp?target=”	ภายใน tag <a> … โดยจะมี tag และ คร่อมโดย ต้องตัดส่วนของ HTML tag อื่นทิ้งไป
Lycos	String เริ่มต้นจะอยู่หลังจาก keyword “Matching web page”	ภายใน tag <a> … โดยจะมี tag <p><a ซึ่งข้อมูลจะอยู่ในบรรทัดถัดไป และปิดท้ายด้วย

สำหรับ User Interface เพื่อการให้บริการควรเป็น interface ที่ง่ายในการใช้ เพราะต้องการเพียงช่องใส่ข้อความคำค้นและปุ่มสำหรับการส่งคำร้องขอบริการ

สถาปัตยกรรมและองค์ประกอบของระบบ

ในการออกแบบระบบ ได้ออกแบบสถาปัตยกรรมและองค์ประกอบของระบบแสดงได้ดังรูปที่ 3.1 และรูปที่ 3.2

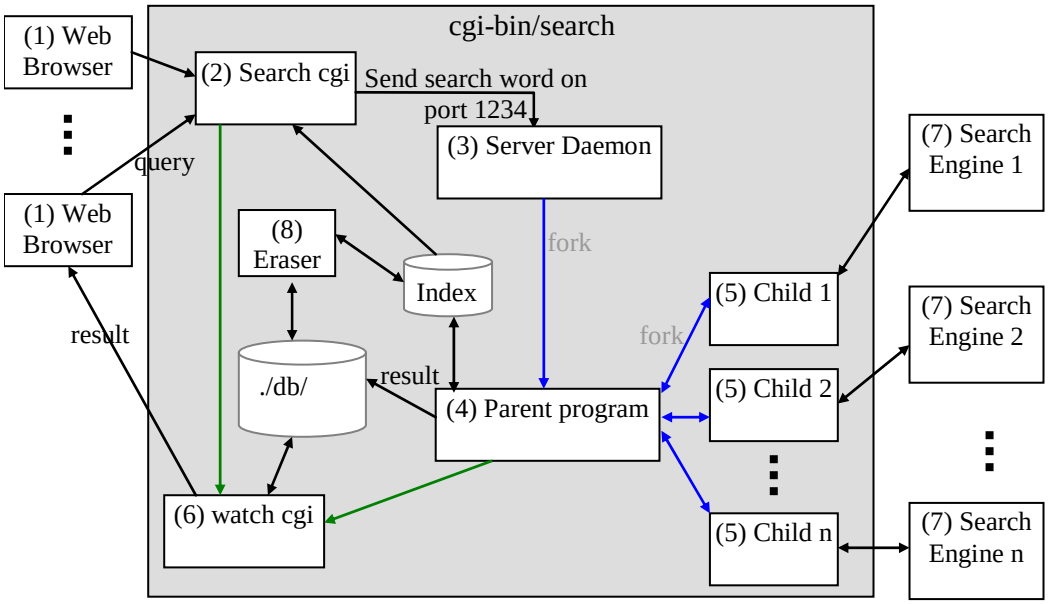


รูปที่ 3.1 Architecture ของระบบ Multiple Search Engine

จากรูปที่ 3.1 แสดงสถาปัตยกรรมของระบบ ซึ่งเป็นระบบ Client/Server โดยประกอบด้วย Web Browser, Multiple Search Engine และ General Purposed Search Engines โดย Multiple Search Engine ทำหน้าที่เป็น web server เพื่อรอรับการค้นคำจากการร้องขอของ web browser จากนั้นส่งไปยัง General Purposed Search Engine ที่เลือกไว้ รอรับผลส่งกลับจาก Search Engine เพื่อนำมาประมวลผลโดยแปลงและจัดรูปแบบคำตอบใหม่ พร้อมกำจัดความซ้ำซ้อนอันเนื่องมาจากคำตอบซ้ำกันจาก search engine อื่น จากนั้นส่งคำตอบที่เรียบเรียงใหม่กลับไปยัง web browser

โดย Multiple Search Engine มีองค์ประกอบของ Process และไฟล์ข้อมูลที่เกี่ยวข้องแสดงดัง รูปที่ 3.2 และแต่ละองค์ประกอบทำหน้าที่โดยสรุปดังนี้

1. ส่วนของ clients ซึ่งเป็น Web Browser ที่จะเป็น interface ให้กับผู้ใช้ในการกรอกคิวรีหรือคำค้นที่ต้องการค้นหา และเป็นส่วนแสดงผลลัพธ์หรือคำตอบที่ได้จากการค้นหาด้วย
2. ส่วนของโปรแกรม client search cgi ทำหน้าที่ในการรับคำที่ต้องการสืบค้น ทำการตรวจสอบกับ cache โดยตรวจกับ index file ว่าคำค้นนี้เคยสืบค้นมาแล้วหรือไม่
    a. ถ้าหากเคยมีปรากฏใน cache ทำการเรียก watch cgi เพื่อให้เปิดไฟล์แล้วอ่านคำตอบส่งกลับไปยัง client ได้เลยโดยตรง
    b. ในกรณีที่เป็นคำค้นใหม่ ส่งคำค้นนี้ไปยัง server daemon โดยใช้พอร์ต 1234 เพื่อให้ดำเนินการค้นผ่าน General Purposed Search Engines



รูปที่ 3.2 องค์ประกอบของระบบ Multiple Search Engine

3. ส่วน Server daemon เมื่อได้รับคำค้น จะทำการ fork parent process เพื่อให้ประมวลผลบริการการค้นคำ สำหรับตัว server daemon เองเมื่อสร้าง parent process นี้ เรียบร้อยแล้ว ก็จะกลับไปรอรับการร้องขอจากผู้ใช้รายอื่น ๆ ต่อไป
4. ส่วนของ Parent process จะทำการ fork child processes ย่อยที่เหมาะสมสำหรับแต่ละ Search Engine เพื่อทำการแปลงคิวรีแล้วส่งไปให้ Search Engine เฉพาะนั้น จากนั้นรอคำตอบจาก child processes ซึ่งจะทำหน้าที่รวบรวมคำตอบที่ได้ไปเก็บไว้ในไฟล์ โดยต้องตรวจสอบลำดับไฟล์ปัจจุบันเพื่อหาลำดับใหม่ของไฟล์ที่จะเขียนไว้เป็น cache ของคำตอบ
5. สำหรับแต่ละ child process ของแต่ละ search engine จะนำคำค้นมาปรับแก้เป็นคิวรีที่เหมาะสมสำหรับการส่งคำร้องต่อไปยัง General Purposed Search Engine (ซึ่งได้แก่ www.altavista.com, www.lycos.com, www.hotbot.com) ที่พอร์ต 80 จากนั้นรอผลลัพธ์กลับ เพื่อส่งต่อคำตอบกลับไปยัง parent process
6. watch cgi ทำหน้าที่อ่านผลลัพธ์จากไฟล์โดยตรวจสอบชื่อไฟล์จาก index จากนั้นส่งผลคำตอบกลับไปยัง client
7. ส่วนของ Search Engine ซึ่งเป็นส่วนที่เก็บข้อมูลเกี่ยวกับ web site และมีระบบ information retrieval ที่สมบูรณ์ของตนเอง ในส่วนนี้จะเป็น source (แหล่งข้อมูล) ให้กับระบบ Multiple Search Engine
8. eraser ทำหน้าที่ในการ clean up cache ที่หมดอายุ โดยจะตื่นขึ้นมาตรวจสอบทุกชั่วโมง ไฟล์ใดที่มีอายุครบ 1 วันจะลบไฟล์นั้นทิ้งไป

จากสถาปัตยกรรมและองค์ประกอบข้างต้น พร้อม Search Engine ที่ถูกเลือกให้เป็นแหล่งข้อมูลให้กับระบบ 3 ตัวคือ AltaVista, Hotbot, และ Lycos ซึ่งในแต่ละ Search Engine มีวิธีการค้นหาที่ใกล้เคียงกันคือรองรับคิวรีที่เป็นลักษณะของนิพจน์ Boolean แต่ผลลัพธ์ของแต่ละ engine นั้นแตกต่างกันออกไปในเรื่องของจำนวนและรูปแบบดังที่กล่าวไว้แล้วในหัวข้อก่อนหน้านี้

การทำ Implementation

เครื่องมือที่ใช้ในการพัฒนาระบบ

- Web Server
- ระบบปฏิบัติการ Linux
- gcc compiler

โดยโค้ดของโปรแกรมทั้งหมด (Version ภาษาซีบนระบบปฏิบัติการ Linux) มีรายละเอียดอยู่ในภาคผนวก 2

เนื่องจาก Search Engine และเทคโนโลยีมีการเปลี่ยนแปลงไป ทำให้โค้ดเดิมที่ถูกพัฒนาดังกล่าวไม่สามารถทำงานได้ในปัจจุบัน (พ.ศ. 2550) ผู้วิจัยจึงได้พัฒนาเป็นเวอร์ชันใหม่ที่ทำงานบนระบบปฏิบัติการ Windows และเปลี่ยนเครื่องมือพัฒนาเป็น Java แทน

สำหรับเครื่องมือที่ใช้ในการพัฒนาระบบ ในเวอร์ชันใหม่ใช้

- Web Server: Apache Tomcat เวอร์ชัน 6.0
- ระบบปฏิบัติการ Window XP
- ภาษา Java (J2SE 5.0)
- ระบบจัดการฐานข้อมูล MS Access 2003

โดยระบบมีสถาปัตยกรรมคล้ายกับระบบที่พัฒนาไว้เดิม แต่เปลี่ยนการเก็บข้อมูลลงไฟล์เป็นการเก็บข้อมูลลงระบบจัดการฐานข้อมูลแทน โดยมีรายละเอียดการทำงานในแต่ละส่วนที่เปลี่ยนแปลงไปดังนี้

การสกัดเลือกเนื้อหาจาก Search Engine

เนื่องจาก Search Engine ที่ใช้เป็นแหล่งข้อมูลมีการเปลี่ยนแปลง ดังนั้นจากการทดสอบการค้นหาบน Search Engine ต่าง ๆ พบว่า Search Engine มีการเปลี่ยนแปลงรูปแบบการค้นหาและการทำงานไปดังตารางที่ 33 โดยทั้ง HotBot และ Lycos ในปัจจุบันใช้ Ask.com Search Engine จึงให้ผลลัพธ์เหมือนกัน ทำให้ต้องเปลี่ยน Hotbot เป็น Google Search Engine [8] ซึ่งปัจจุบันได้รับความนิยมสูงสุดแทน

ตารางที่ 33 แพทเทิร์น String สำหรับการคิวรี (2550)

Search Engine	**คิวรี String**
AltaVista	`http://www.altavista.com/web/results?itag=ody&q=%s HTTP/1.1\n`
HotBot	`http://www.hotbot.com/?query=%s HTTP/1.1\n`
Lycos	`http://search.lycos.com/?query=%s HTTP/1.1\n`
Google	`http://www.google.com/search?&q=%s HTTP/1.1\n`

สำหรับแพทเทิร์นในการตัด String เพื่อหาคำตอบแสดงดังตารางที่ 3.4

ตารางที่ 3.4 แพทเทิร์นการตัด String เพื่อหาจุดเริ่มต้นและหา Link ของคำตอบ

Search Engine	**จุดเริ่มแรกของ URL คำตอบ**	**ข้อมูลคำตอบ**
AltaVista	String เริ่มต้นจะอยู่หลังจาก tag “`</DIV>` `<br class='lb'>”`	อยู่ระหว่าง tag `<a ...> ... </a>` โดยจะต้องตัดส่วนของ HTML tag ทิ้งไป
Hotbot	String เริ่มต้นจะอยู่หลังจาก `</div><ul><li><div>`	ภายใน tag `<a ...> ... </a>` โดยจะมี tag `</li>` บอกจุดสิ้นสุดของ link
Lycos	String เริ่มต้นจะอยู่หลังจาก keyword `<ol class="rslts_Lst" start="1"> <li>`	ภายใน tag `<a> ... </a>` โดยจะมี tag `</li>` บอกจุดสิ้นสุดของ link
Google	String เริ่มต้นจะอยู่หลังจาก `<h2 class=r>`	ภายใน tag `<a ...> ... </a>` โดยจะต้องตัดส่วน tag อื่น ก่อน `<a href=` ทิ้งไป

และเพื่อให้สามารถรองรับการเปลี่ยนแปลงในเรื่องของรูปแบบที่จะใช้ในการดึงคำตอบที่ส่งมาจากแต่ละ Search Engine ซึ่งมีแพทเทิร์นในผลลัพธ์แตกต่างกัน ในงานวิจัยนี้เลือกใช้การจับคู่โดย Regular Expression โดยจะเก็บ Profile ของรูปแบบไว้ในไฟล์ และเมื่อเริ่มต้นจะอ่านค่าเข้ามาจากไฟล์แพทเทิร์นสำหรับแต่ละ Search Engine แทนการเขียนโค้ดแบบฝังตายตัวไว้ในโปรแกรม สำหรับรูปแบบของไฟล์จะกำหนดแพทเทิร์นสำหรับสิ่งที่ต้องใช้ในการดึงข้อมูลไว้ใน 2 บรรทัด โดยบรรทัดแรกบอกชนิด และบรรทัดที่ 2 บอกรูปแบบ สำหรับชนิดที่ใช้ในงานนี้กำหนดไว้ 4 ประเภทหลักคือ

- #found เป็นแพทเทิร์นในกรณีที่คำค้นนั้นมีคำตอบ
- #top เป็น top level ของแต่ละ item คำตอบหรือเป็นทั้งหมดของ item ในคำตอบ
- #attribute ระบุถึงแพทเทิร์นของ anchor ของ link และคำอธิบายของแต่ละ item คำตอบ

- #next ระบุถึง anchor หรือ URL สำหรับการร้องขอหน้าถัดไปของคำตอบ

ตัวอย่างของ Profile สำหรับ Google แสดงดังรูปที่ 3.3

```
#top
#Pattern=\<h2 class=r>(.*?\</table>)
#attribute
#Pattern=\<a href="([^"]*)"[^>]*>(.*?)\</a>\</h2>(.*?\<br>)\<span
#next
#Pattern=\<a href=([^>]*)>\<div id=nn>\</div>Next\</a>
#found
#Pattern=Results.*?of\sabout\s\<b>([0-9,]*?)</b>.*?\<div id=res>
```

1) แต่ละ item ของผลลัพธ์ หรือ กลุ่มคำตอบ
2) Anchor ของ แต่ละ item ของผลลัพธ์
3) หัวเรื่องสำหรับส่วน anchor ของผลลัพธ์
4) คำอธิบายโดยย่อสำหรับ anchor นั้น
5) URL ของ next page ในการร้องขอส่วนคำตอบในหน้าถัดไป
6) จำนวนที่พบ

รูปที่ 3.3 แสดงตัวอย่าง Profile ของ Google Search Engine

ในแต่ละแพทเทิร์นจะกำหนดสิ่งที่ดึงออกมาเพื่อใช้งานไว้ใน ( ) และในแต่ละกลุ่มของค่าที่ดึงออกมาได้ จะถูกนำใช้เป็นข้อมูลผลลัพธ์ และเป็นแหล่งต้นตอข้อมูลในหน้าถัดไปของ Search Engine

ตัวอย่าง การใช้แพทเทิร์นในรูปที่ 3.3 กับข้อมูลบางส่วนของ HTML ที่ได้จาก Google

ในส่วนนี้แสดงตัวอย่างบางส่วนของผลลัพธ์ที่ได้จาก Google Search Engine ในการค้นคำ “สุรยุทธ์ จุลานนท์” เพื่อแสดงตัวอย่างของสิ่งที่ดึงมาเพื่อนำมาสร้างเป็นผลลัพธ์ของ Multiple Search Engine โดยแต่ละ item คำตอบแสดงด้วยตัวเอียง และรูปแบบที่เป็นตัวกำหนดแสดงด้วยตัวหนาและขีดเส้นใต้

1. บางส่วนของ HTML ที่บอกผลลัพธ์การค้นหาจาก Google

<table cellpadding=0 cellspacing=0 border=0><tr><td> </td></tr><tr><td height=7><img width=1 height=1 alt=""></td></tr></table></td></form></tr></table><table border=0 cellpadding=0 cellspacing=0 width=100% class="t bt"><tr><td nowrap> Web </td><td align=right nowrap>**Results** 1 - 10 **of about ***222,000***** for สุรยุทธ์ จุลานนท์. (0.20 seconds) </td></tr></table>** ** <h2 class=r>

:

2. ผลลัพธ์จากการตัดข้อมูลโดยใช้แพทเทิร์น #found ที่กำหนดสำหรับ Google

222,000

3. บางส่วนของเนื้อหาใน HTML ที่ได้จาก Google

<table border=0 cellpadding=0 cellspacing=0 width=100% class="t bt"><tr><td nowrap> Web </td><td align=right nowrap>**Results 1 - 10 of about 222,000 for สุรยุทธ์ จุลานนท์.** (0.20 seconds) </td></tr></table> **<h2 class=r>**สุรยุทธ์ จุลานนท์ - วิกิพีเดีย</h2><table border=0 cellpadding=0 cellspacing=0><tr><td class=j>พลเอก สุรยุทธ์ จุลานนท์ เกิดที่จังหวัดเพชรบุรี เป็นบุตรของพันโทพโยม จุลานนท์ (บุตร พันเอกพระยาวิเศษสิงหนาถ (ยิ่ง จุลานนท์)) ...
th.wikipedia.org/wiki/<wbr>สุรยุทธ์_จุลานนท์ - 106k - <nobr>Cached - Similar pages</nobr></td></tr>**</table>** **<h2 class=r>**<a

:

4. ผลลัพธ์จากการตัดข้อมูลโดยใช้แพทเทิร์น #attribute ที่กำหนดสำหรับ Google

โดยที่จะได้ข้อมูลแยกออกเป็น 3 ส่วนคือ ส่วนที่ 1) anchor ของ hyperlink ของคำตอบ ส่วนที่ 2) ข้อความแสดง (title) และส่วนที่ 3) คำอธิบายโดยย่อของคำตอบนั้น

ส่วนที่ 1 :

http://th.wikipedia.org/wiki/%E0%B8%AA%E0%B8%B8%E0%B8%A3%E0%B8%A2%E0%B8%B8%E0%B8%97%E0%B8%98%E0%B9%8C_%E0%B8%88%E0%B8%B8%E0%B8%A5%E0%B8%B2%E0%B8%99%E0%B8%99%E0%B8%97%E0%B9%8C

ส่วนที่ 2 :

สุรยุทธ์ จุลานนท์ - วิกิพีเดีย

ส่วนที่ 3 :

<table border=0 cellpadding=0 cellspacing=0><tr><td class=j><font size=-1>????? <table border=0 cellpadding=0 cellspacing=0><tr><td class=j><font size=-1>พลเอก สุรยุทธ์ จุลานนท์ เกิดที่จังหวัดเพชรบุรี เป็นบุตรของพันโทพโยม จุลานนท์ (บุตร พันเอกพระยาวิเศษสิงหนาถ (ยิ่ง จุลานนท์)) ...

5. บางส่วนของ HTML สำหรับการหา Next จาก Google

ในส่วนนี้จะใช้ค้นหาตัวอ้างอิงถึงหน้าถัดไปของข้อมูลผลลัพธ์จาก Search Engine หรือที่ปรากฏทั่วไปในหน้าคำตอบในรูปของ Next

*N> 9<td nowrap> 10<td nowrap class=b><**a href=**/search?q=%E0%B8%AA%E0%B8%B8%E0%B8%A3%E0%B8%A2%E0%B8%B8%E0%B8%97%E0%B8%98%E0%B9%8C+%E0%B8%88%E0%B8%B8%E0%B8%A5%E0%B8%B2%E0%B8%99%E0%B8%99%E0%B8%97%E0%B9%8C&hl=en&rlz=1T4GFRC_enTH214TH219&start=10&sa=N>** Next**

:

6. ผลลัพธ์จากการตัดข้อมูลโดยใช้แพทเทิร์น #next ที่กำหนดสำหรับ Google

ข้อมูลที่ตัดได้จากแพทเทิร์น #next จะใช้เป็นตัวกำหนด URL ของการค้นหาต่อ

/search?q=%E0%B8%AA%E0%B8%B8%E0%B8%A3%E0%B8%A2%E0%B8%B8%E0%B8%97%E0%B8%98%E0%B9%8C+%E0%B8%88%E0%B8%B8%E0%B8%A5%E0%B8%B2%E0%B8%99%E0%B8%99%E0%B8%97%E0%B9%8C&hl=en&rlz=1T4GFRC_enTH214TH219&start=10&sa=N

สำหรับ Profile ของแพทเทิร์นของ Search Engine ทั้งหมดดูรายละเอียดได้จากภาคผนวก 1

การกำหนด URL สำหรับการค้นหาจากแหล่ง Search Engine ใน mode ต่าง ๆ

ในการค้นหาแบบทั่วไป ใช้ mode การค้นหาจาก search engine แบบกรณีทั่วไป (ดังแสดงไว้ใน **ตารางที่ 33** ก่อนหน้านี้) โดยมี Source 3 แหล่งคือ Google, AltaVista และ Lycos โดยใช้ URL สำหรับการค้นหาโดยสรุปดังนี้คือ

Google: http://www.google.com/search?q=**KEYWORD**

AltaVista: http://www.altavista.com/web/results?itag=ody&q= **KEYWORD**

Lycos: http://search.lycos.com/?query= **KEYWORD**

เมื่อ KEYWORD คือคำที่ต้องการค้นหา โดยคำค้นนี้จะต้องถูกแปลงให้เป็นไปตามข้อกำหนดของ URL ดังที่กล่าวไว้แล้วในบทที่ 2

ในกรณีการค้นแบบ Match ทั้งหมด หรือ exact phrase เลือกใช้การค้นหาใน mode Advanced Search อย่างไรก็ตามเนื่องจาก Lycos ไม่รองรับการค้นหาแบบ exact phrase ดังนั้นหากกำหนดให้ค้นหาแบบ exact phrase จะใช้แหล่งข้อมูลเพียง 2 แหล่งเท่านั้น เนื่องจากคำตอบที่ไม่ตรงกับความต้องการค้นหา โดย URL สำหรับการค้นหาเป็นดังนี้

Google: http://www.google.com/search?as_epq= **"KEY+WORD"**

AltaVista:

http://www.altavista.com/web/results?itag=ody&pg=aq&aqmode=s&aqp=**KEY+WORD**

โดยคำที่ต้องการค้น คือ KEY WORD ซึ่ง Google กำหนดให้ใส่ " " คร่อมคำที่ต้องการค้นด้วย

การรวมข้อมูลจากหลาย Search Engine

ก่อนที่จะรวมข้อมูลคำตอบจากหลายแหล่งเข้าด้วยกัน จะทำการตรวจสอบการซ้ำของคำตอบที่ได้จาก Search Engine ในงานวิจัยนี้ใช้ตรวจสอบโดยอาศัยเทคนิค Hashing ในกรณีที่พบค่าซ้ำจะไม่เพิ่มรายการคำตอบลงในฐานข้อมูล แต่จะปรับปรุงค่าของ engine ว่าเป็นแหล่งของคำตอบนั้นด้วย

หลังจากได้ข้อมูลที่ต้องการแล้ว ในการรวมข้อมูลจากหลาย Search Engines ในระบบนี้เลือกรูปแบบการนำเสนอเหมือนกันกับรูปแบบที่ใช้โดยทั่วไปในการแสดงผลของ Search Engine กล่าวคือ แสดง Hyperlink พร้อมข้อความ title ของ link นั้น เนื้อหาโดยย่อ และ URL อย่างย่อของแหล่งข้อมูลคำตอบ และรูปแบบของ HTML กำหนดไว้ดังต่อไปนี้

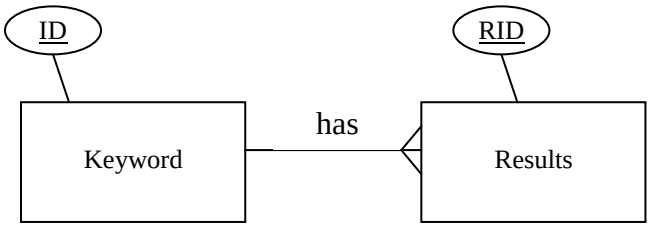
```
<HTML><HEAD><TITLE>Multiple Search Engine</TITLE>
<meta http-equiv="content-type" content="text/html;
charset=UTF-8"/></HEAD>
<style><!--
div{color:#000}
   .a, .grnLnk{color:green}
   .s{color: black; font-size: 80%}
   .headlines {background:yellow; font-weight:bold;}
   .blacktext {color:black}
--></style>
</HEAD>
<BODY><CENTER>
   <FORM  action="/search" method="get">
         <p><img src="mse4.gif" border="0" align="middle" />
         <input type="text" name="q" size="50"/>
         <input name="go" type="submit" id="go" value="Go" />
         <br><label><input name="e" type="checkbox" />
         exact phrase</label>
         <input type="hidden" name="new" value="new"/>
         </p>
   </FORM>
   </CENTER><H2>Results for: KEYWORD</H2>
<font color='red'>G</font>oogle: ???? <font color='yellow'>
L</font>ycos: ???? <font color='blue'>A</font>ltavista: ????

<ol start='1' type='1'>
<LI><a href="HYPERLINK">TITLE</a><br/>
DESCRIPTION<br/>   From: <font color='red'>G</font><br/>
<span class="a">BRIEF-HYPERLINK</span>
</li><p/>
:
<p><a href=/search?=sessionID>Next</a></p>
</BODY>
</HTML>
```

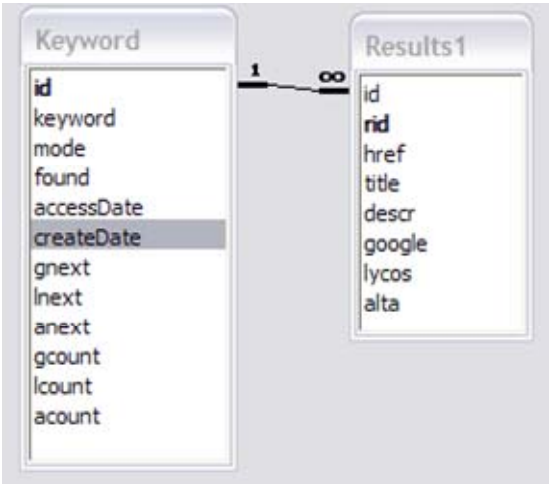
เมื่อเนื้อหาของข้อมูลผลลัพธ์อยู่ในส่วนตัวเข้มและขีดเส้นใต้ โดยเป็นส่วนข้อมูลที่ได้มาจากข้อมูลคำตอบของ Search Engine และมีระบุตัวย่อ G, L และ A เพื่อบ่งบอกแหล่งที่ให้ข้อมูลรายการนั้น ข้อมูลในฟิลด์ **????** เป็นคำตอบค่าสูงสุดของรายการที่ค้นพบจาก Search Engine นั้น ๆ

ระบบฐานข้อมูลที่ใช้ในการเก็บผลลัพธ์

ระบบฐานข้อมูลที่ใช้ในการเก็บผลลัพธ์ ในที่นี้ใช้ MS Access แทนการเก็บลงไฟล์ โดยประกอบด้วย Entity หลัก 2 ตัว ดังแสดงในรูปที่ 3.4 และรูปที่ 3.5 แสดง Physical model ของข้อมูลที่เก็บในระบบจัดการฐานข้อมูล MS Access โดยเมื่อแปลงเป็นตารางแล้วมีรายละเอียดของตารางหลักทั้ง 2 ตารางดังในตารางที่ 3.5 และตารางที่ 3.6 ตามลำดับ



รูปที่ 3.4 Entity-Relationship Model ของข้อมูลที่เก็บโดยแสดงเพียงคีย์หลักของ Entity



รูปที่ 3.5 Physical Data Model ของข้อมูลที่เก็บในระบบจัดการฐานข้อมูล MS Access

ตารางที่ 3.5 แสดง Data Dictionary ของตาราง Keyword ที่ใช้ในระบบ

Field	**Type**	**Description**	**Constraints**
ID	Long Integer	รหัสที่ไม่ซ้ำกันของคำค้น	PK
Keyword	VARCHAR(255)	คำค้น	
Mode	Byte	ระบุ mode การค้นของคำ 0= แบบทั่วไป และ 1=exact phrase	
Found	Boolean	Flag ระบุการค้นว่ามีคำตอบหรือไม่ Yes=มี และ No= ไม่มี	
accessDate	Date	วันและเวลาที่ Access ครั้งสุดท้าย	Default เป็นวันที่

Field	Type	Description	Constraints
			ปัจจุบัน
createDate	Date	วันและเวลาที่สร้างผลลัพธ์ของคำค้น ใช้ช่วยในการกำหนดเวลาหมดอายุ	Default เป็นวันที่ปัจจุบัน
gNext	VARCHAR(255)	Next reference ก่อนหยุดการค้นหาของ google	
lNext	VARCHAR(255)	Next reference ก่อนหยุดการค้นหาของ lycos	
aNext	VARCHAR(255)	Next reference ก่อนหยุดการค้นหาของ altaVista	
gcount	Long Integer	จำนวนสูงสุดที่คืนจาก Google ของคำค้นนี้	Default เป็น 0
lcount	Long Integer	จำนวนสูงสุดที่คืนจาก Lycos ของคำค้นนี้	Default เป็น 0
acount	Long Integer	จำนวนสูงสุดที่คืนจาก AltaVista ของคำค้นนี้	Default เป็น 0

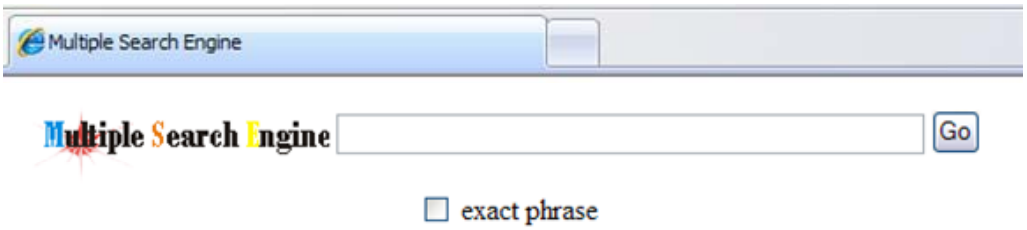
ตารางที่ 3.6 แสดง Data Dictionary ของตาราง Results1 ที่ใช้ในระบบ

Field	Type	Description	Constraints
ID	Long Integer	รหัสของคำค้น	FK
rid	Autonumber	เลขที่ของคำตอบ	PK
href	VARCHAR(255)	Hyperlink ของคำตอบ	
title	VARCHAR(127)	คำ title เมื่อแสดง Hyperlink ของคำตอบ	
descr	VARCHAR(255)	คำอธิบายโดยย่อของคำตอบ	
google	Integer	หมายเลขลำดับของผลลัพธ์จาก Google	
lycos	Integer	หมายเลขลำดับของผลลัพธ์จาก Lycos	
alta	Integer	หมายเลขลำดับของผลลัพธ์จาก AltaVista	

บทที่ 4

ผลการวิจัย (Result)

จากผลการพัฒนาทำให้ได้ Multiple Search Engine ที่สามารถช่วยผู้ใช้ในการค้นหาได้ โดยที่รูปที่ 4.1 แสดงหน้าจอหลักสำหรับการค้นหา



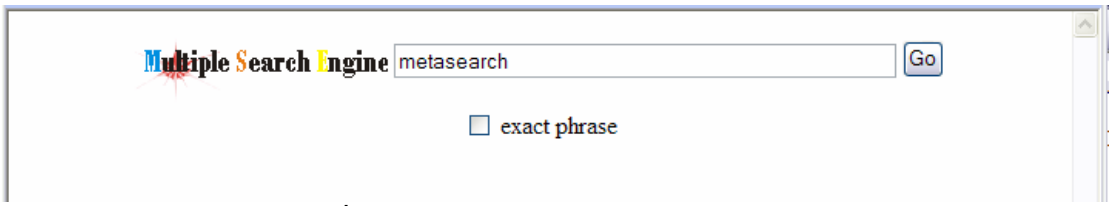
รูปที่ 4.1 แสดงหน้าจอหลักสำหรับการค้นหา

และได้ทดสอบในการค้นคำในรูปแบบต่าง ๆ ดังนี้

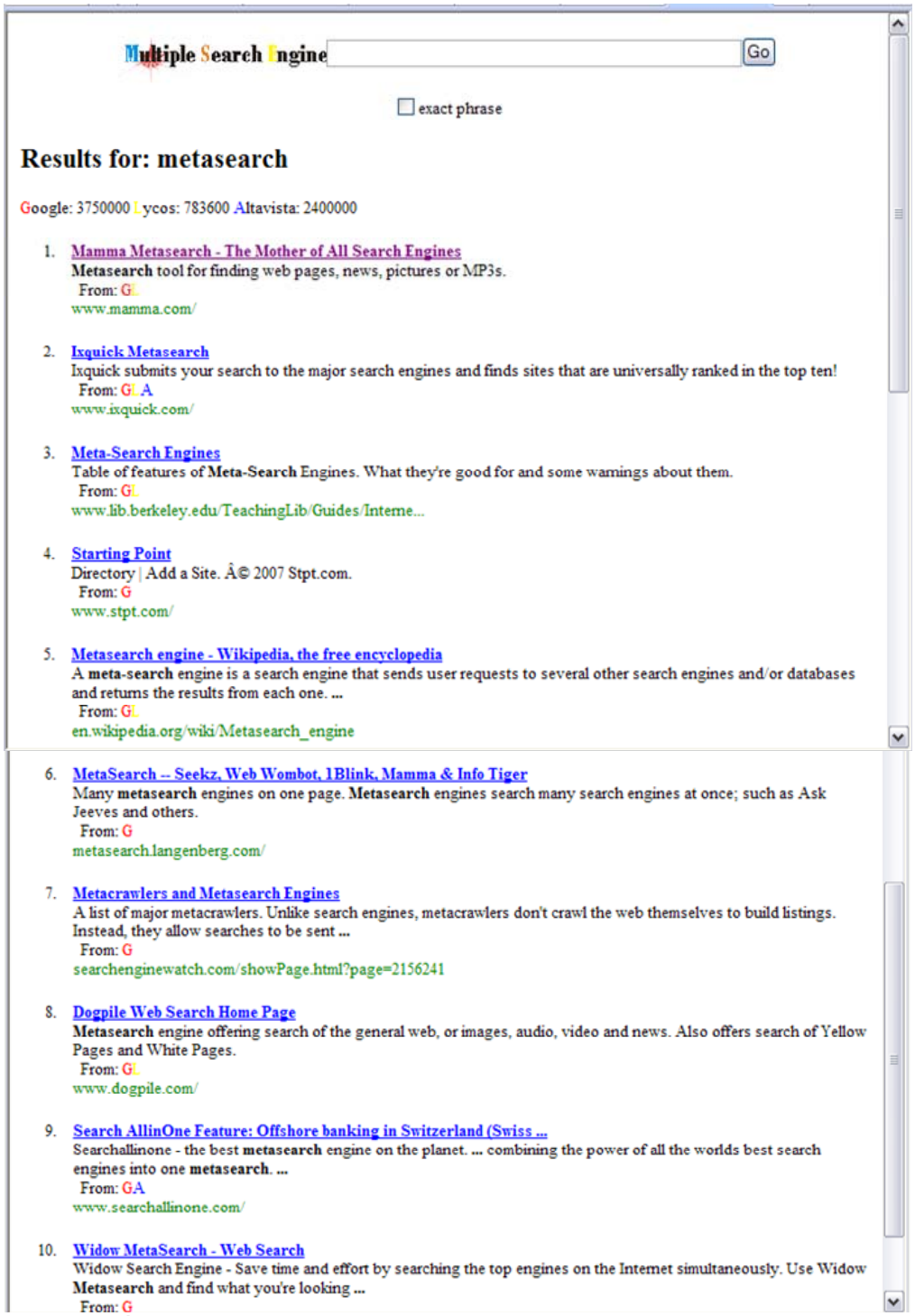
การทดสอบการค้นคำจาก Multiple Search Engine

การค้นคำเดียว

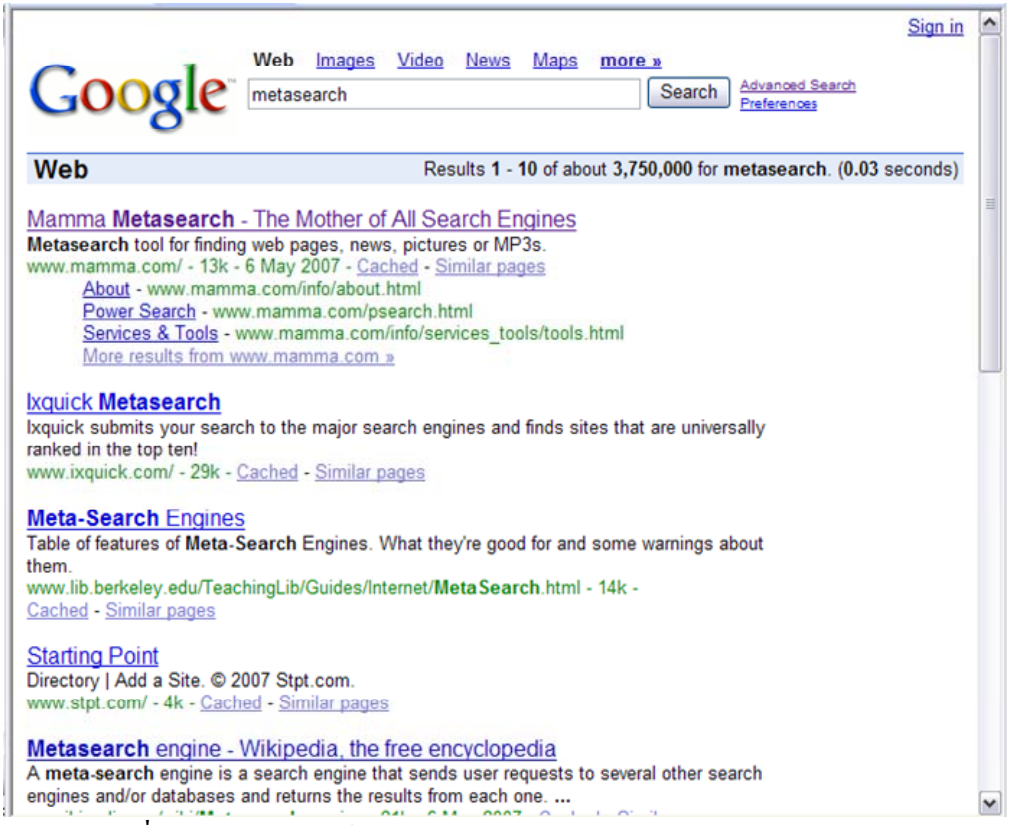
เมื่อผู้ใช้ใส่คำค้นเป็นคำเดียวเช่นคำ metasearch ดังแสดงในรูปที่ 4.2 หลังการ search ให้ผลลัพธ์ดังรูปที่ 4.3 ซึ่งหากเปรียบเทียบผลลัพธ์ที่ได้กับแต่ละ Search Engine ดังแสดงในรูปที่ 4.4 - รูปที่ 4.6 จะเห็นว่า Multiple Search Engine ให้ผลลัพธ์ที่มีคำตอบที่หลากหลายอันเป็นผลลัพธ์ที่ได้จากหลาย Search Engine รวมกัน โดยแต่ละรายการแสดงแหล่งที่มาของรายการนั้น เช่น G=Google, L=Lycos และ A=AltaVista ในกรณีที่ผลลัพธ์มีทั้ง GLA แสดงว่ามาจากทั้ง 3 แหล่งเป็นต้น



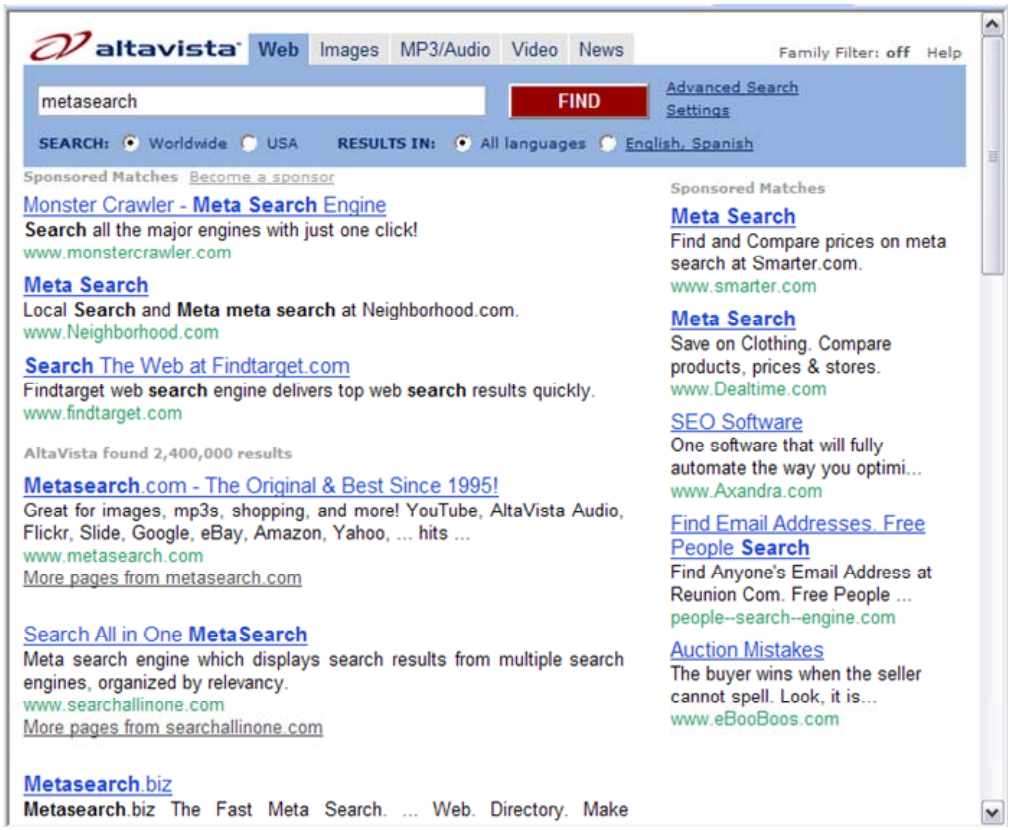
รูปที่ 4.2 แสดงการค้นคำ *metasearch* โดยผู้ใช้



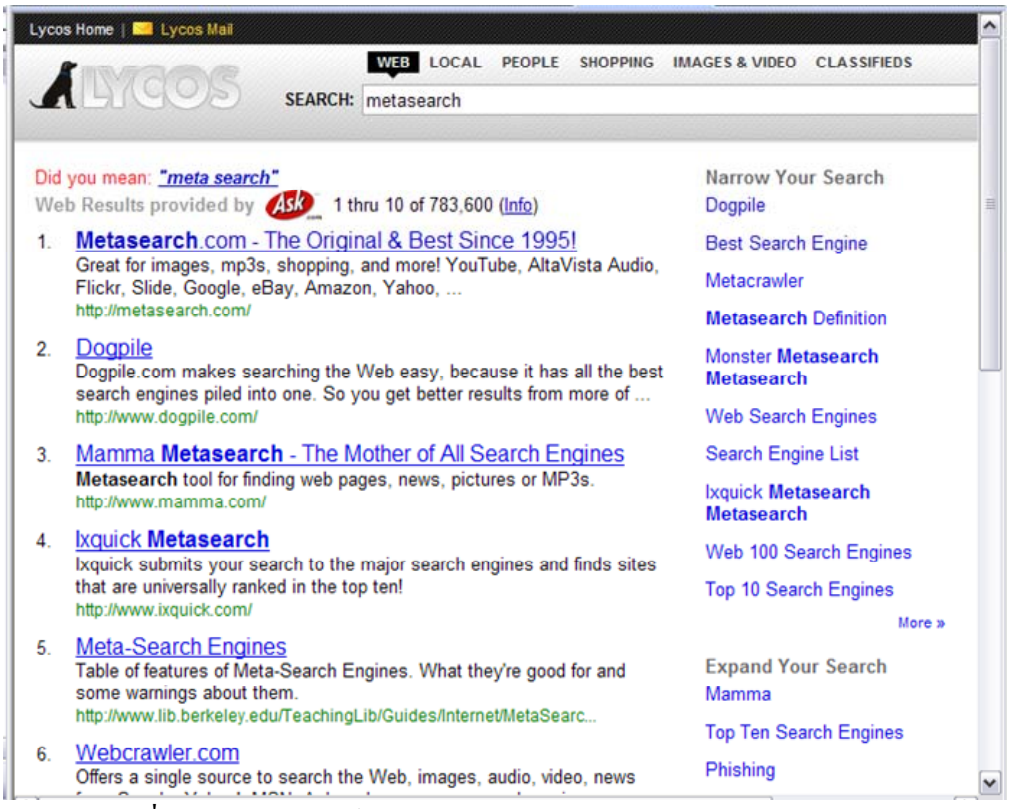
รูปที่ 4.3 แสดงผลลัพธ์การค้นคำ *metasearch* จาก Multiple Search Engine



รูปที่ 4.4 แสดงผลลัพธ์การค้นคำ *metasearch* จาก Google Search Engine



รูปที่ 4.5 แสดงผลลัพธ์การค้นคำ *metasearch* จาก AltaVista Search Engine

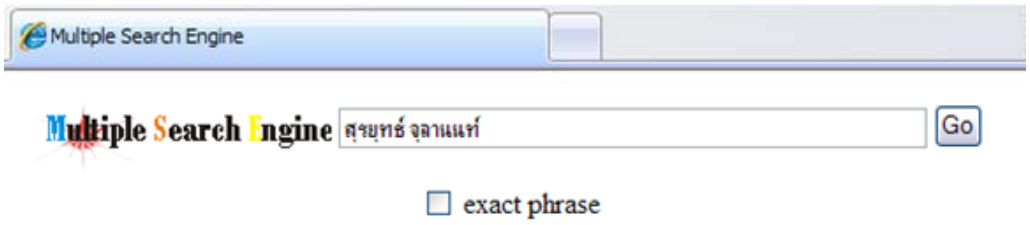


รูปที่ 4.6 แสดงผลลัพธ์การค้นคำ *metasearch* จาก Lycos Search Engine

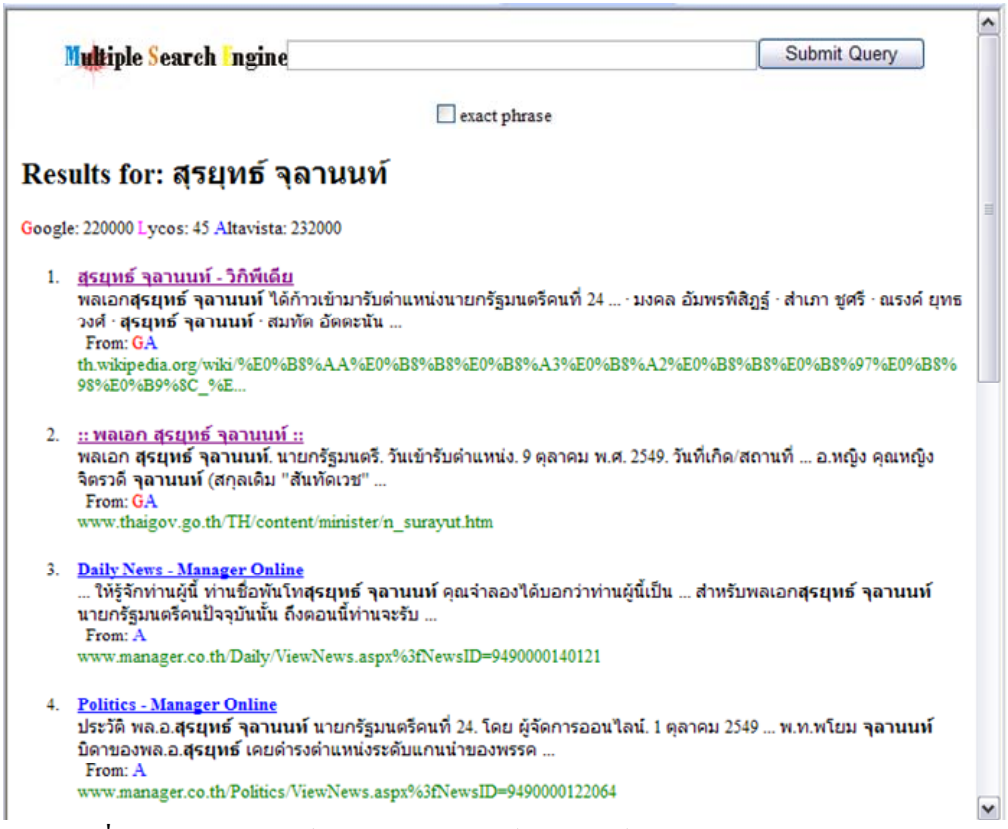
การค้นคำภาษาไทย

ตัวอย่างในรูปที่ 4.7 เป็นตัวอย่างการค้นคำภาษาไทย ซึ่งหลังการ search ให้ผลลัพธ์ดังรูปที่ 4.8 ซึ่งหากเปรียบเทียบผลลัพธ์ที่ได้กับแต่ละ Search Engine ดังแสดงในรูปที่ 4.9 - รูปที่ 4.11

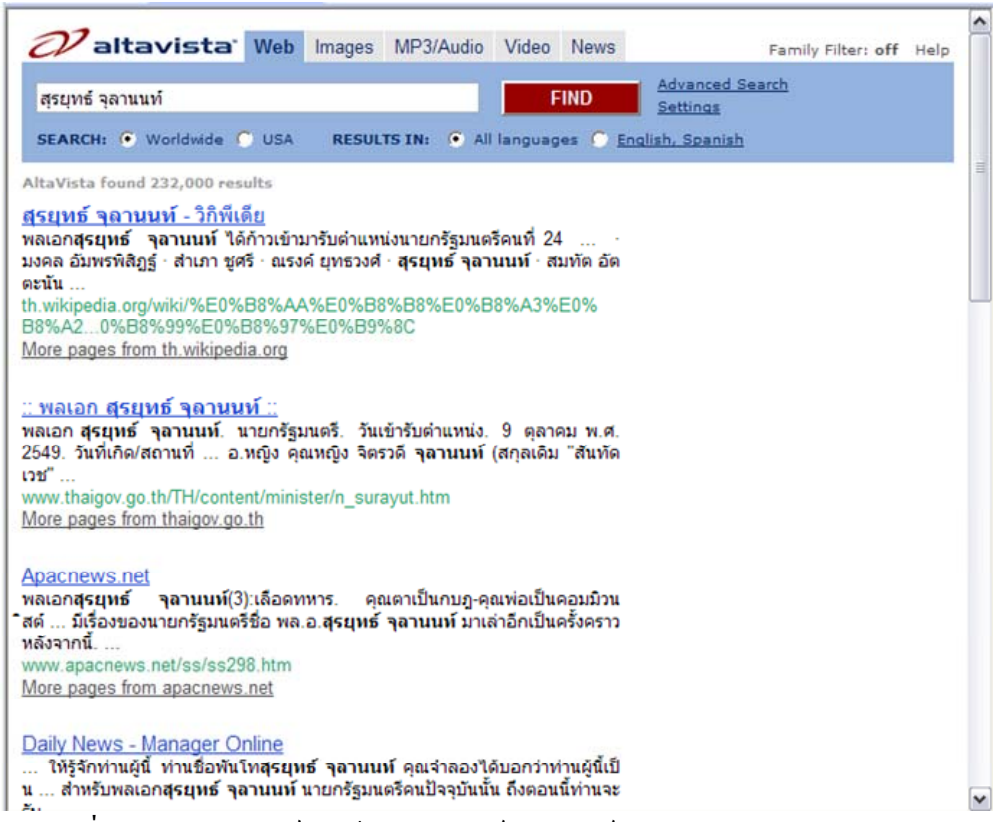
ข้อสังเกตเกี่ยวกับคำตอบจาก AltaVista ในรูปที่ 4.12และ Lycos ในรูปที่ 4.13จะเห็นว่าหากใช้รหัสภาษา (Encoding) ตอนเริ่มต้นค้นคำผิด (เช่น Western (ISO) แทน UTF-8) จะได้ผลลัพธ์การค้นไม่ตรงกับความต้องการการค้นหา อันเนื่องมาจากการถอดรหัสคำค้นเริ่มต้นที่ผิดพลาด โดยสังเกตได้จากช่องคำค้นของ Search Engine ที่ระบุเป็นรหัสไม่ตรงกับคำค้นที่ต้องการจริง



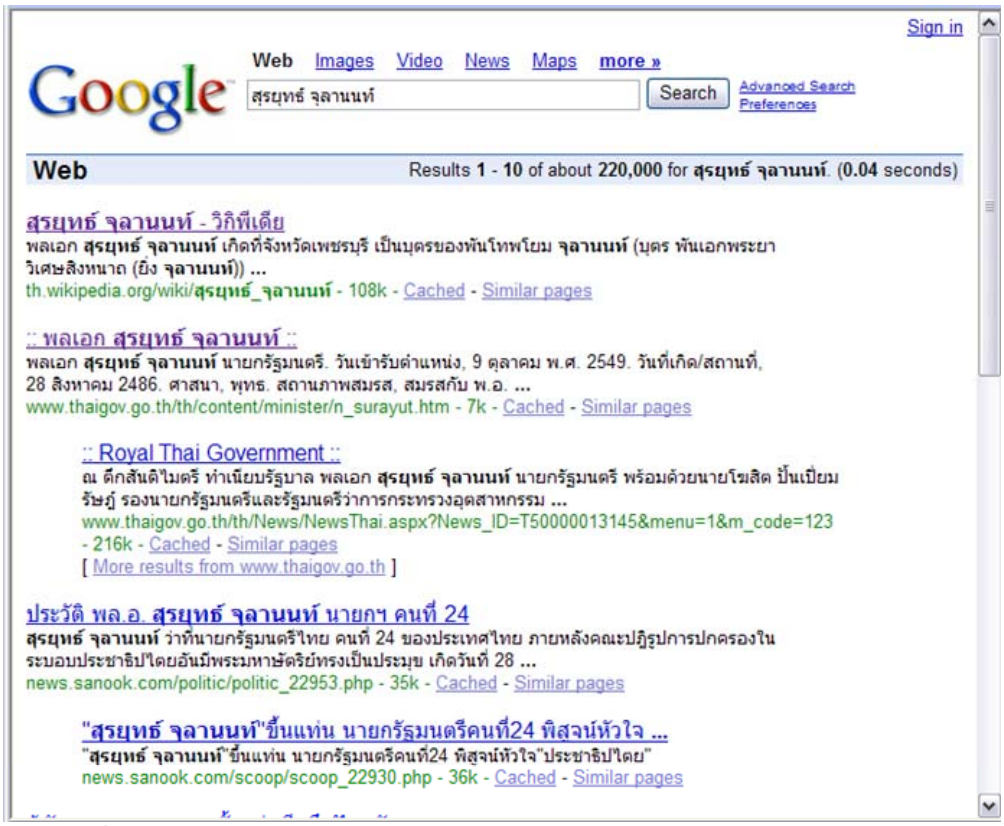
รูปที่ 4.7 แสดงการค้นคำ *สุรยุทธ์ จุลานนท์* โดยผู้ใช้



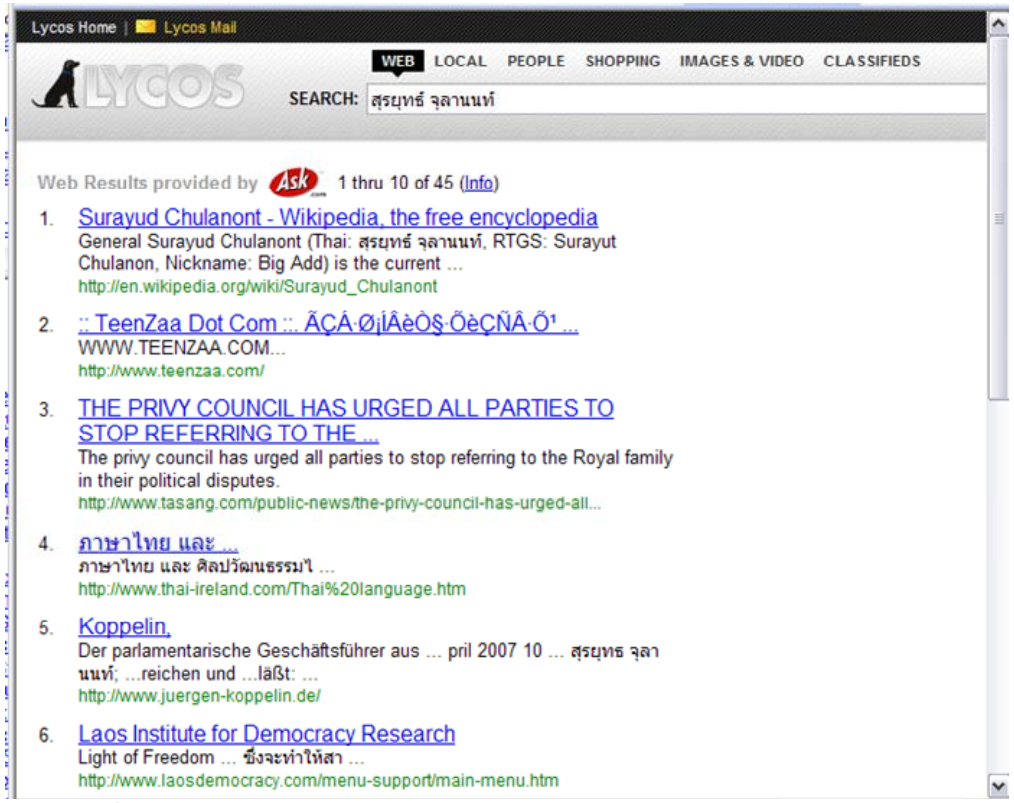
รูปที่ 4.8 แสดงผลลัพธ์การค้นคำ *สุรยุทธ์ จุลานนท์* จาก Multiple Search Engine



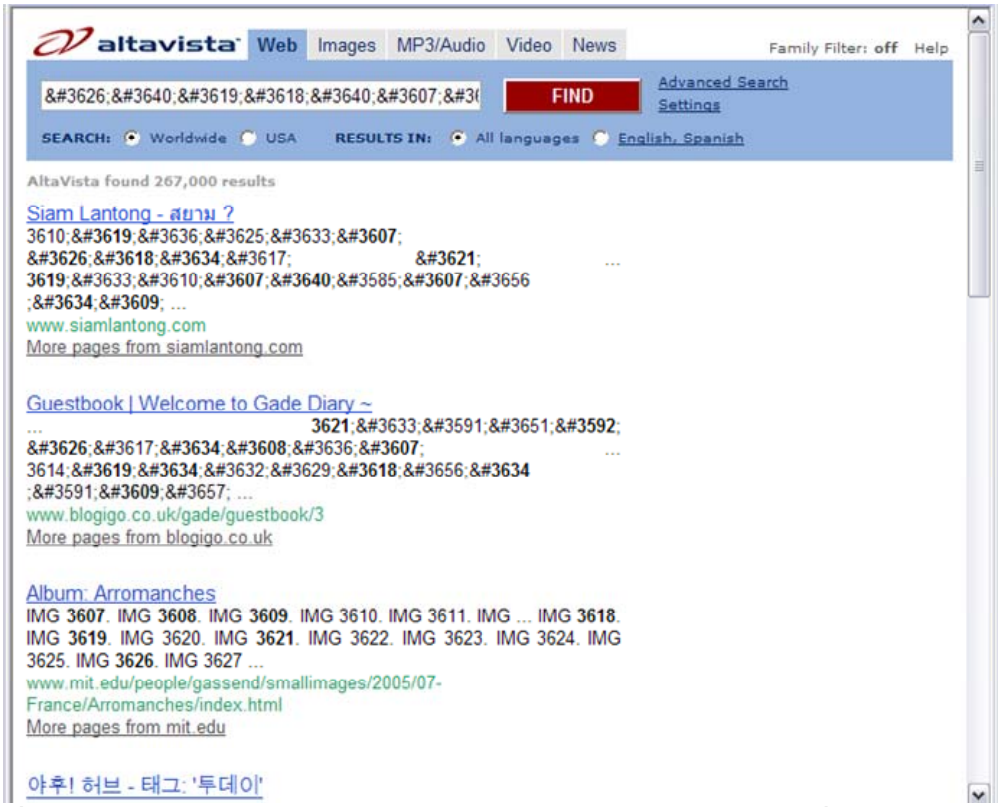
รูปที่ 4.9 แสดงผลลัพธ์การค้นคำ *สุรยุทธ์ จุลานนท์* จาก AltaVista Search Engine



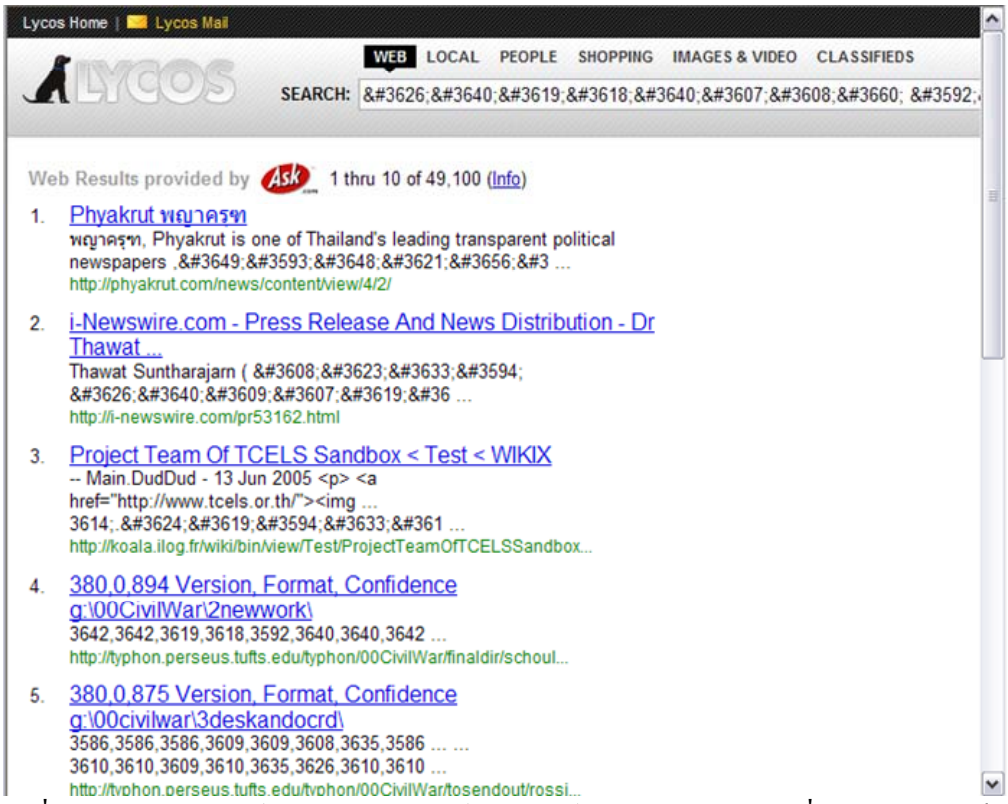
รูปที่ 4.10 แสดงผลลัพธ์การค้นคำ *สุรยุทธ์ จุลานนท์* จาก Google Search Engine



รูปที่ 4.11 แสดงผลลัพธ์การค้นคำ *สุรยุทธ์ จุลานนท์* จาก Lycos Search Engine



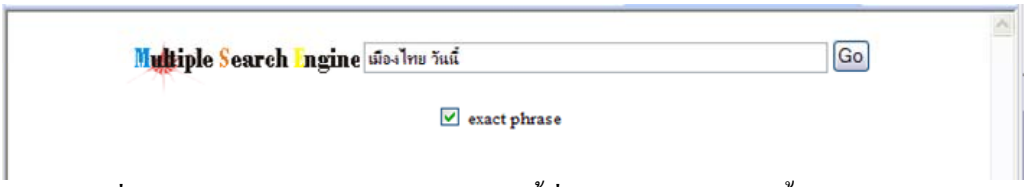
รูปที่ 4.12 แสดงผลลัพธ์การค้นคำ *สุรยุทธ์ จุลานนท์* จาก AltaVista ในกรณีที่ใช้ Encoding เป็น Western (ISO) ในตอนค้นคำ



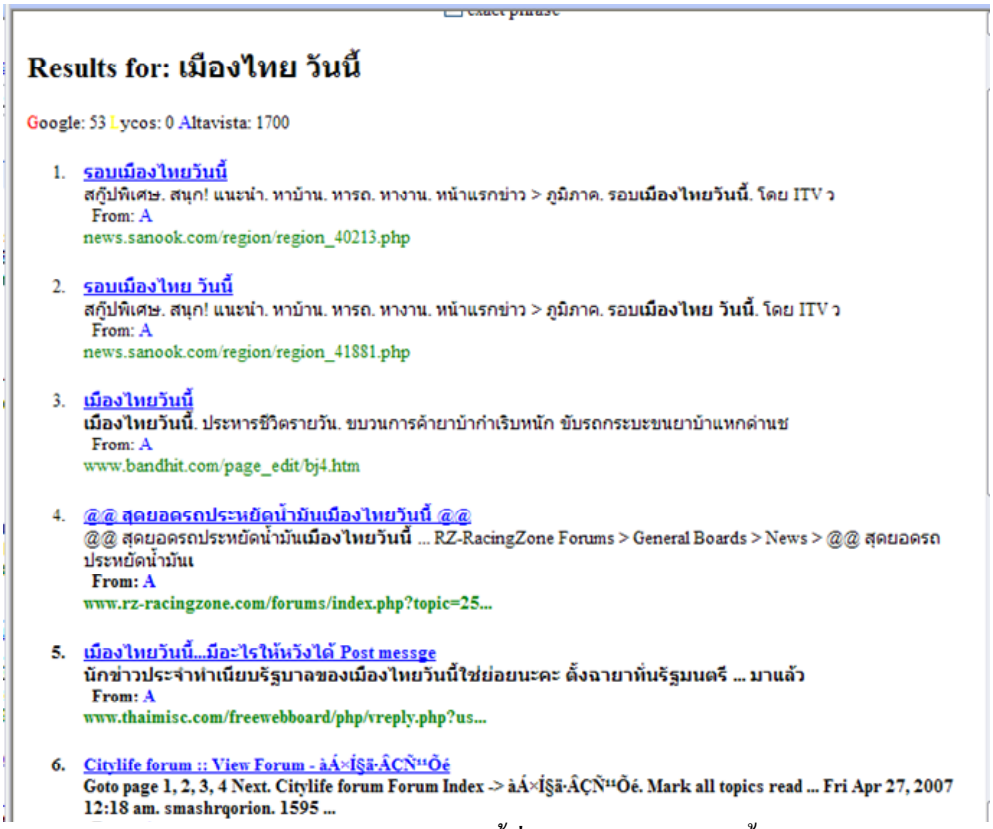
รูปที่ 4.13 แสดงผลลัพธ์การค้นคำ *สุรยุทธ์ จุลานนท์* จาก Lycos ในกรณีที่ใช้ Encoding เป็น Western (ISO) ในตอนค้นคำ

การค้นคำแบบ exact phrase

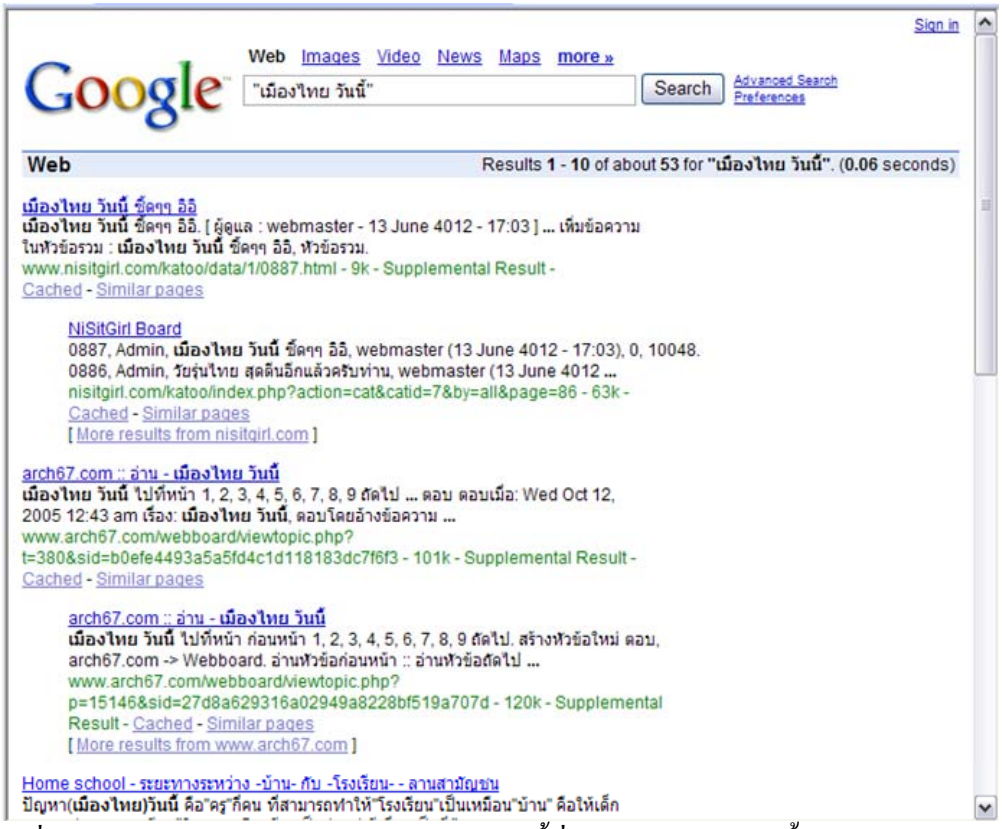
เมื่อผู้ใช้ใส่คำค้นเป็นคำที่ต้องการให้ match ทั้งคำ (รูปที่ 4.14) หลังการ search ให้ผลลัพธ์ดังรูปที่ 4.15 โดยในกรณีนี้จะไม่ส่งคำค้นไปยัง Lycos เนื่องจาก Lycos ไม่รองรับการค้นหาคำแบบ exact phrase และเมื่อเปรียบเทียบคำตอบที่ได้จาก Google และ AltaVista Search Engine ดังรูปที่ 4.16 และรูปที่ 4.18 ตามลำดับ ในรูปที่ 4.17 และรูปที่ 4.19 เป็นการเปรียบเทียบกับกรณีที่ค้นหาคำแบบทั่วไปของ Google และ AltaVista ตามลำดับ จะเห็นว่า กรณีที่ Match exact phrase Google ให้คำตอบเพียง 53 รายการ แต่ถ้าเป็นกรณีทั่วไปจะให้คำตอบทั้งสิ้น 24,300 รายการ และ AltaVista ให้รายการประมาณ 1,700 รายการ สำหรับกรณีทั่วไปจะได้ 625,000 รายการ



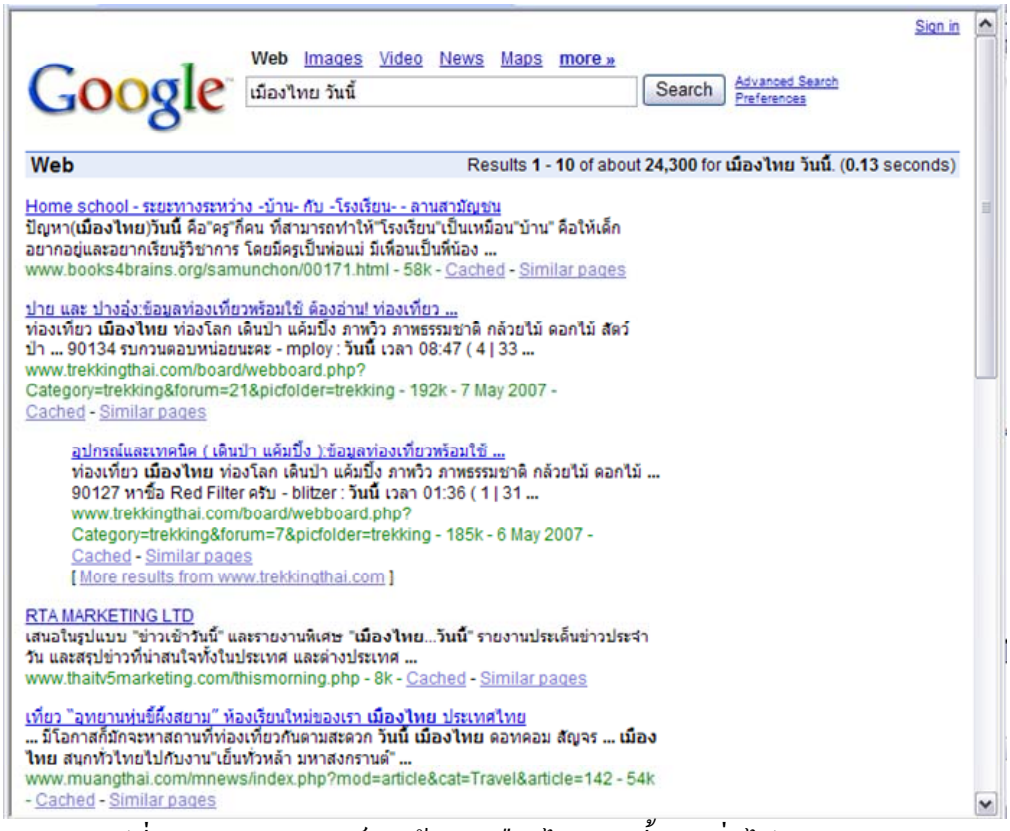
รูปที่ 4.14 แสดงการค้นคำ *เมืองไทย วันนี้* ที่ต้องการให้ Match ทั้งหมดโดยผู้ใช้



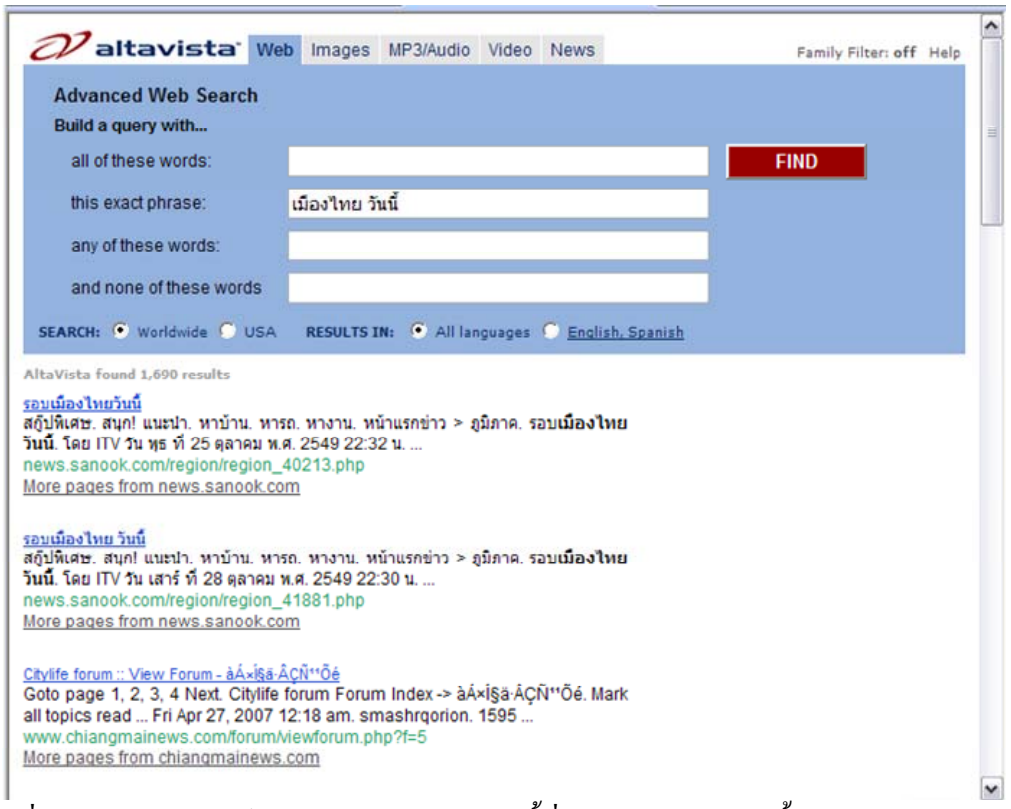
รูปที่ 4.15 แสดงผลลัพธ์การค้นคำ *เมืองไทย วันนี้* ที่ต้องการให้ Match ทั้งหมด (exact phrase) ของ Multiple Search Engine



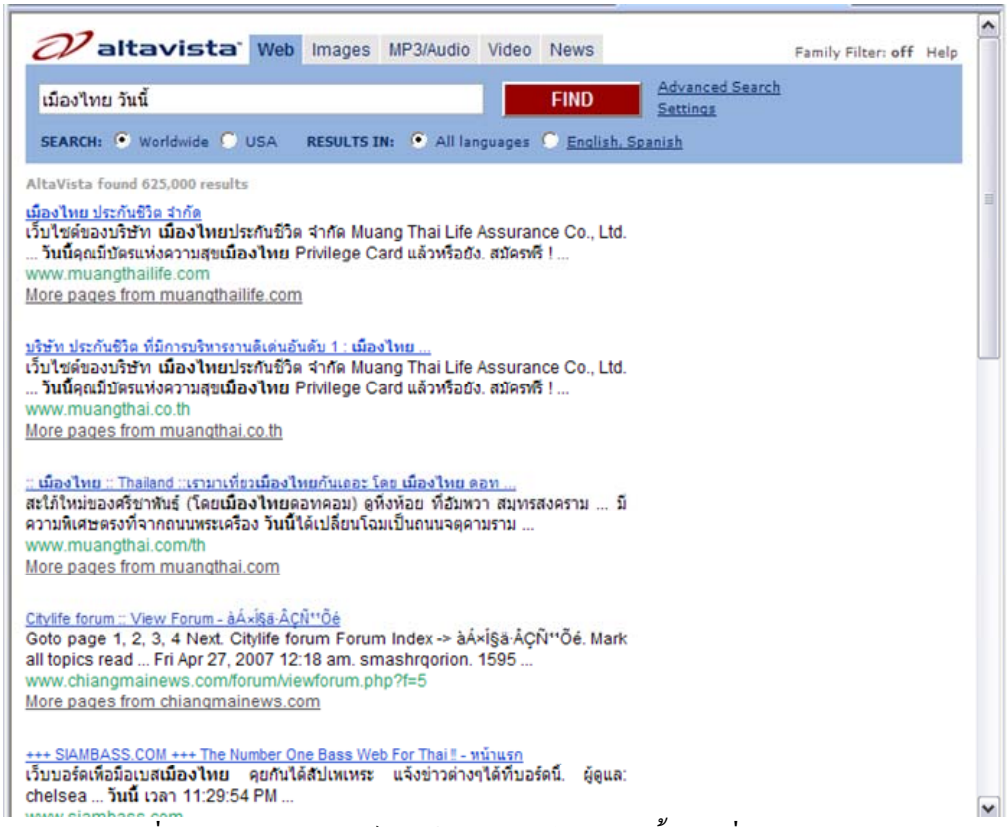
รูปที่ 4.16 แสดงผลลัพธ์การค้นคำ *เมืองไทย วันนี้* ที่ต้องการให้ Match ทั้งหมด จาก Google



รูปที่ 4.17 แสดงผลลัพธ์การค้นคำ *เมืองไทย วันนี้* แบบทั่วไปจาก Google

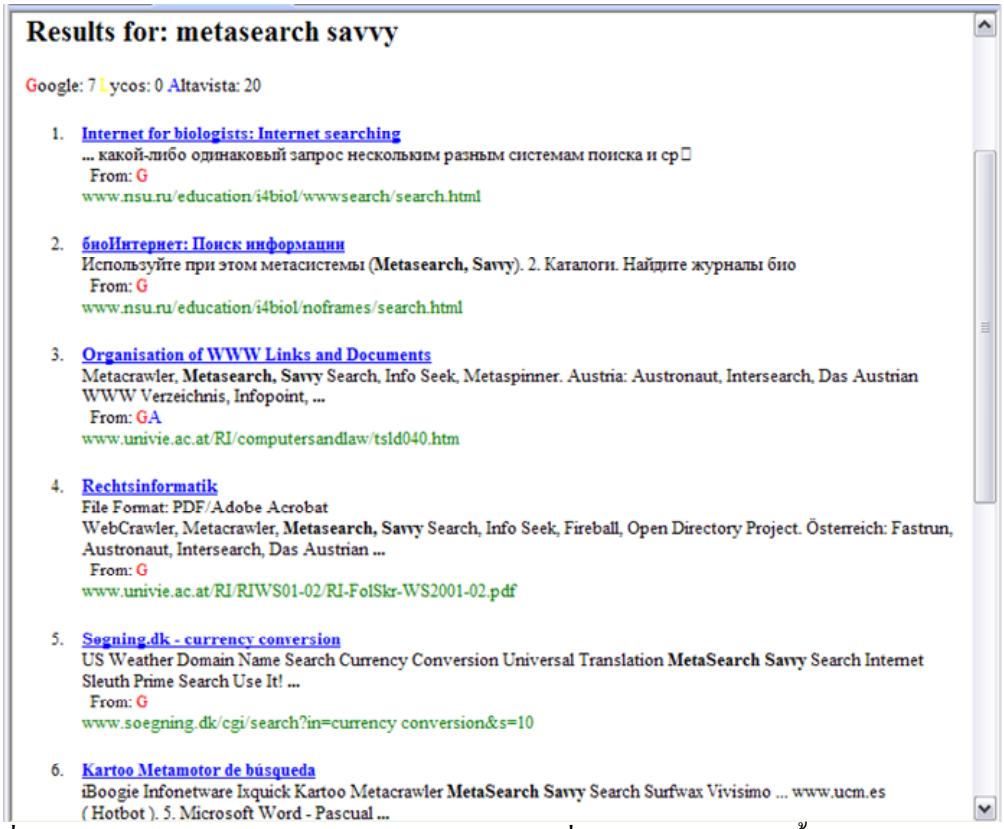


รูปที่ 4.18 แสดงผลลัพธ์การค้นคำ *เมืองไทย วันนี้* ที่ต้องการให้ Match ทั้งหมด จาก AltaVista

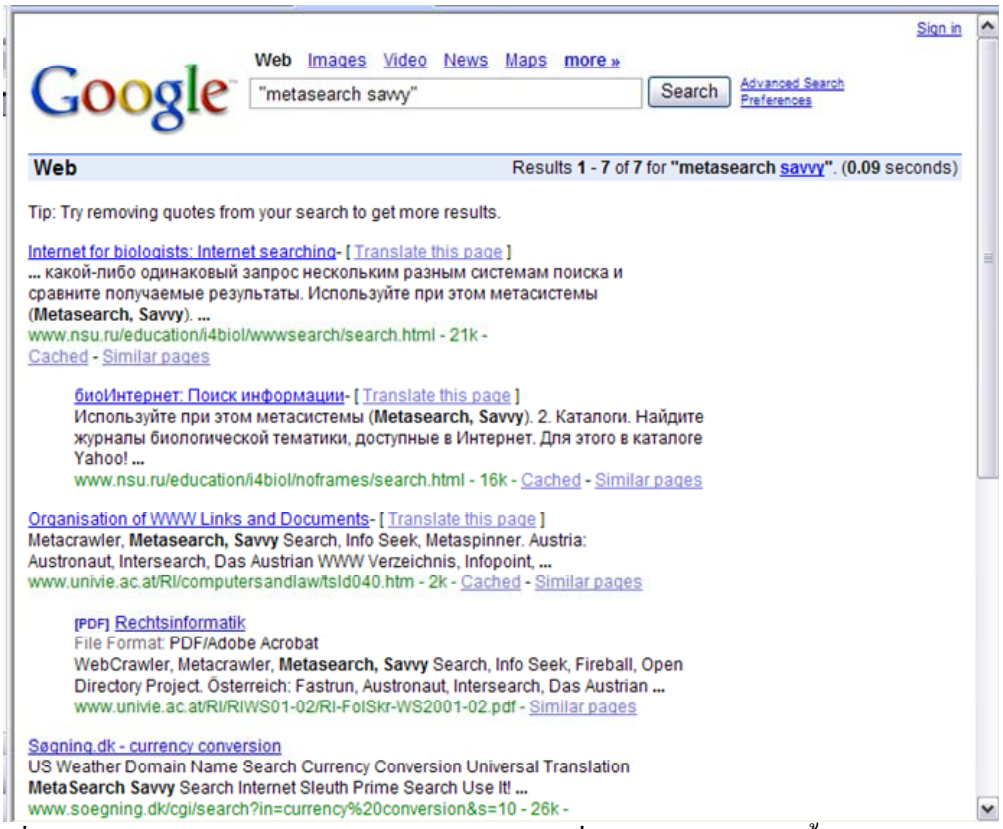


รูปที่ 4.19 แสดงผลลัพธ์การค้นคำ *เมืองไทย วันนี้* แบบทั่วไปจาก AltaVista

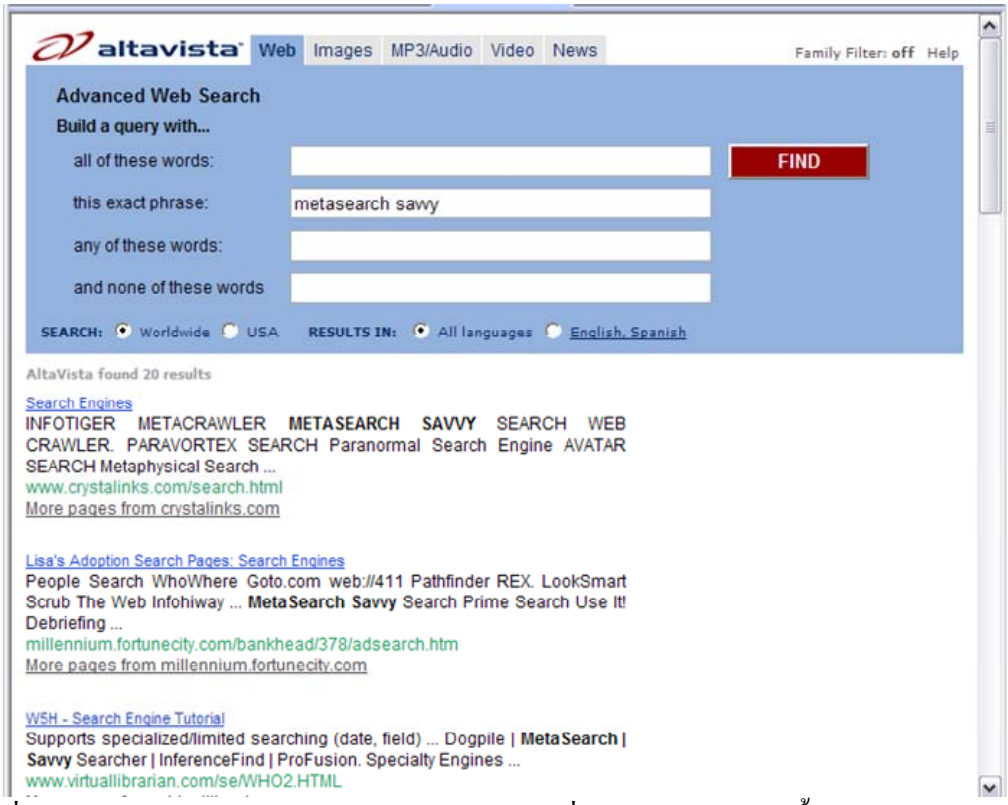
อีกตัวอย่างในการค้นแบบให้ match ทั้งคำ โดยเมื่อผู้ใช้ใส่คำค้น metasearch savvy ผลลัพธ์การค้นแสดงดังรูปที่ 4.20 และเมื่อเปรียบเทียบคำตอบที่ได้จาก Google และ AltaVista Search Engine ดังรูปที่ 4.21 และรูปที่ 4.22 ตามลำดับ จะเห็นว่าลำดับของผลลัพธ์เป็นผลจาก Google ก่อน AltaVista เนื่องจากลำดับการเรียงคำตอบขึ้นกับความเร็วในการตอบจาก Search Engine



รูปที่ 4.20 แสดงผลลัพธ์การค้นคำ *metasearch savvy* ที่ต้องการให้ Match ทั้งหมด (exact phrase) ของ Multiple Search Engine



รูปที่ 4.21 แสดงผลลัพธ์การค้นคำ *metasearch savvy* ที่ต้องการให้ Match ทั้งหมด จาก Google

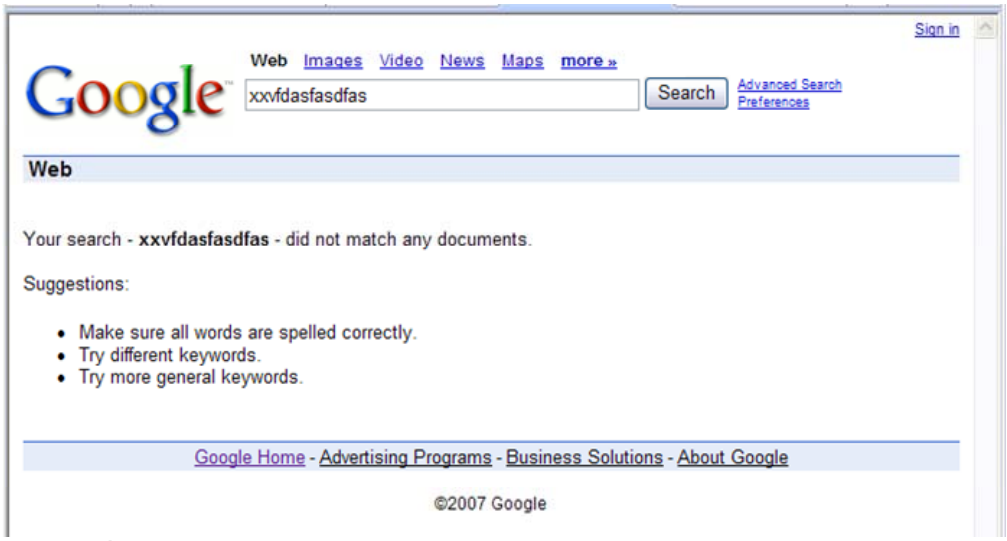


รูปที่ 4.22 แสดงผลลัพธ์การค้นคำ metasearch savvy ที่ต้องการให้ Match ทั้งหมด จาก AltaVista

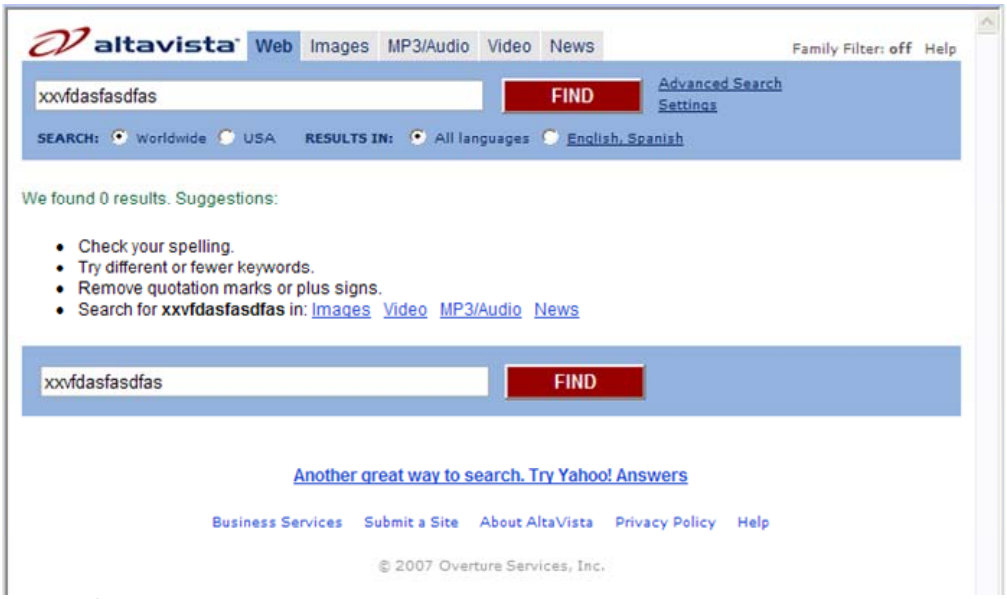
การค้นคำโดยไม่พบคำตอบ

รูปที่ 4.23 แสดงตัวอย่างกรณีการค้นคำ xxvfdasfasdfas แต่ไม่พบคำตอบ และเมื่อค้นคำเดียวกันจาก Google, AltaVista และ Lycos Search Engine จะได้ดังรูปที่ 4.24 - รูปที่ 4.26 ตามลำดับ

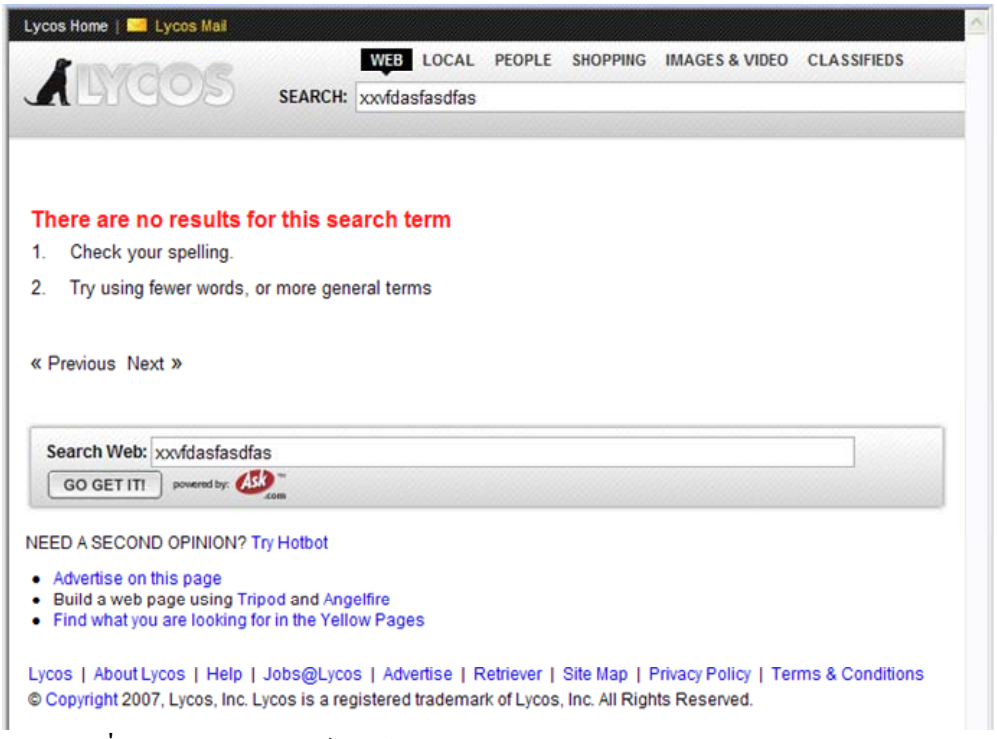
รูปที่ 4.23 แสดงผลลัพธ์การค้นคำ xxvfdasfasdfas แต่ไม่พบคำตอบของ Multiple Search Engine



รูปที่ 4.24 แสดงผลลัพธ์การค้นคำ xxvfdasfasdfas แต่ไม่พบคำตอบของ Google



รูปที่ 4.25 แสดงผลลัพธ์การค้นคำ xxvfdasfasdfas แต่ไม่พบคำตอบของ AltaVista



รูปที่ 4.26 แสดงผลลัพธ์การค้นคำ xxvfdasfasdfas แต่ไม่พบคำตอบของ Lycos

บทที่ 5

อภิปรายผล (Discussion)

การสร้างเครื่องมือช่วยในการค้นหาคำค้นจาก Search Engine หลายแหล่ง เป็นทางเลือกสำหรับผู้ใช้ในการค้นหาโดยไม่ต้องศึกษาการใช้เครื่องมือของแต่ละ Search Engine และไม่จำเป็นต้องจดจำ URL ของแต่ละ Search Engine ทั่วไป นอกจากนี้ยังไม่จำเป็นต้องสลับเปลี่ยนหน้าเว็บเพจเพื่อการค้นหาจากหลายแหล่งในเวลาเดียวกัน ช่วยสร้างความสะดวกในการใช้งาน และยังเหมาะกับผู้ใช้ที่ใหม่ต่อการค้นหาบนเว็บจาก Search Engine ทั่วไป

ในการพัฒนา พบว่าแต่ละ Search Engine มีรูปแบบการค้นไม่เหมือนกัน และมีความสามารถไม่เท่าเทียมกัน ดังนั้นการรวมความสามารถที่หลากหลายของแต่ละ Search Engine เข้าด้วยกันจะช่วยเพิ่มความสะดวก เพิ่มความครอบคลุมและลดขั้นตอนการเรียนรู้ของผู้ใช้ได้

ข้อสังเกตที่พบในการค้นคำภาษาไทย คือการเริ่มต้นคำค้นจำเป็นต้องใช้การเข้ารหัสภาษาให้ถูกต้อง เพราะการถอดรหัสคำค้นโดย Search Engine อิงกับรูปแบบของ Encoding ที่เริ่มต้นของการค้นหา สำหรับงานวิจัยนี้เนื่องจากรองรับภาษาไทยและภาษาอังกฤษ ดังนั้นจึงใช้การเข้ารหัสเป็น UTF-8 เป็นกรณีปริยาย ทำให้คำตอบที่ได้ตรงกับภาษาไทยที่ตั้งใจค้น อย่างไรก็ตามหากจะให้เครื่องมือรองรับภาษาต่าง ๆ ซึ่งมีการเข้ารหัสที่แตกต่างกันไปจำเป็นต้องปรับเปลี่ยนการเข้ารหัสใหม่เพื่อให้มีความถูกต้อง

เนื่องจากการสร้าง Multiple Search Engine ขึ้นอย่างมากกับรูปแบบการส่งคำค้นและรูปแบบผลลัพธ์ที่ตอบกลับมาจาก Search Engine ดังนั้นเมื่อใดก็ตามที่ Search Engine มีการปรับปรุงไม่ว่าจะเป็นส่วนวิธีการส่งคำค้นหรือรูปแบบการแสดงผล จะส่งผลกระทบกับการทำงานของ Multiple Search Engine ทันที ดังนั้นสำหรับการกำหนดรูปแบบการส่งคำค้นไปยัง Search Engine จึงแยกออกเป็น Profile สำหรับแต่ละ Search Engine นั้น ๆ ซึ่งสร้างความสะดวกในการปรับเปลี่ยน ถึงแม้ Profile นี้จะง่ายในการปรับเปลี่ยน แต่การตรวจสอบว่ามีการเปลี่ยนแปลงในวิธีการค้นหรือไม่ ยังจำเป็นต้องอาศัยมนุษย์เป็นผู้ตรวจสอบและทำการปรับปรุง โดยอาจสังเกตจาก URL ที่ใช้ในการค้นคำของ Search Engine ตัวนั้น ๆ

เช่นเดียวกันกับวิธีการค้น ในการรวมผลลัพธ์จากแต่ละ Search Engine จะแยกเป็น Profile สำหรับสกัดคำตอบ ช่วยให้ง่ายในการปรับเปลี่ยน สำหรับการสกัดคำตอบเพื่อแยกรูปแบบหาจุดเริ่มต้นนั้นอาจไม่ง่ายอย่างการเปลี่ยนรูปแบบคำค้นซึ่งสังเกตได้จาก URL ที่ใช้ส่งคำค้นได้รูปแบบผลลัพธ์ตรวจสอบได้ยากกว่าเนื่องจากไม่อาจสังเกตได้ทันที ต้องดูรายละเอียดจากเนื้อหาที่ได้เป็นคำตอบจากการค้น

ในปัจจุบัน Search Engine บางตัวเช่น Google ได้ให้การสนับสนุนการค้นคำซึ่งจะสะดวกและง่ายขึ้นโดยผ่านบริการ Google Web Service [9] ถึงแม้จะไม่ใช่บริการหลักของ Google เพราะมี

ข้อกำหนดในการใช้ เช่นจำนวนการคิวรีไม่เกิน 1,000 คิวรีต่อวัน การเข้าถึงต้องมี Access Key (ซึ่ง Google ยุติการสนับสนุน SOAP Search API Key ตั้งแต่เดือนธันวาคม 2549 แล้ว) แต่ก็นับเป็นนิมิตรหมายอันดีสำหรับการค้นคำบนอินเทอร์เน็ตที่อาจเพิ่มเติมความสามารถโดยอาศัย Search Engine ที่มีศักยภาพในการค้นคำโดยทั่วไปเพื่อปรับให้เหมาะกับการใช้งานเฉพาะในบางกรณีได้

ถึงแม้การสร้าง Multiple Search Engine จะมีช่วงเวลาการใช้งานได้และยังขึ้นกับรูปแบบต่าง ๆ ของแหล่งข้อมูล แต่ในแง่ของความต้องการการรวมข้อมูลจากหลายแหล่งซึ่งเป็นปัญหาทั่วไปในยุคโลกาภิวัตน์ก็ยังจำเป็นต้องมีการค้นคว้าวิจัยต่อไป

บทที่ 6

สรุปและข้อเสนอแนะ (Conclusion and Recommendation)

สรุปการวิจัย

ในงานวิจัยนี้ มีวัตถุประสงค์เพื่อสร้างเครื่องมือช่วยในการค้นหาคำค้นจากหลายแหล่งได้ในเวลาเดียวกัน โดยผู้ใช้ไม่จำเป็นต้องเรียนรู้ที่ตั้งและวิธีการใช้งานของแต่ละ Search Engine ช่วยสร้างความสะดวกในการใช้งาน Multiple Search Engine ที่สร้างขึ้นสามารถช่วยในการค้นหาคำค้นต่าง ๆ ได้โดยมีความสามารถหลักสรุปได้ดังนี้

1. ใช้แหล่ง Search Engines คือ AltaVista, HotBot และ Lycos (ในเวอร์ชันภาษาซี บนระบบปฏิบัติการ Linux) และได้เปลี่ยนแหล่ง Search Engine เป็น AltaVista, Google และ Lycos เนื่องจาก HotBot และ Lycos ใช้ Ask™ Search Engine [4] ตัวเดียวกันในการค้นหา และ Google เป็น Search Engine ที่ได้รับความนิยมสูงสุดในปัจจุบัน
2. สามารถรวบรวมผลลัพธ์จากแต่ละ Search Engine โดยเรียบเรียงให้เป็นผลลัพธ์บนหน้าเว็บเพจเดียวกัน โดยลำดับที่ปรากฏเรียงตามความรวดเร็วและลำดับที่ได้รับคำตอบจาก Search Engine
3. สามารถค้นคำเดียว หลายคำในรูป OR และคำที่เป็น exact phraseได้
4. สามารถค้นคำภาษาไทยได้
5. มีการตรวจสอบจาก cache หากเป็นการค้นคำคำเดิมที่เคยค้นแล้ว โดยการกำหนดการหมดอายุของคำค้นใช้เวลา 1 วันเป็นตัวกำหนด ซึ่งเป็นการยอมรับ Weak Consistency กับแหล่งข้อมูลจริงในระยะ 1 วัน เนื่องจากแต่ละ Search Engine มีนโยบายในการปรับปรุงข้อมูลไม่เหมือนกันจึงยากจะคาดเดาว่าเมื่อใด Engine ใดมีการเปลี่ยนแปลง การไล่ตรวจสอบแต่ละ Search Engine จะใช้เวลามากกว่าที่จะกำหนดให้ค้นหาใหม่แทน

การประยุกต์ใช้

จากการวิจัยนี้สามารถประยุกต์ใช้แนวคิดเกี่ยวกับการสืบค้นและการ Integrate ข้อมูลจากหลายแหล่งที่มีความแตกต่างกัน ทั้งในเรื่องการตั้งข้อคำถามสำหรับการสืบค้น ลักษณะผลลัพธ์ของคำตอบที่จำเป็นต้องปรับให้มีรูปแบบเดียวกัน แนวคิดจากงานวิจัยนี้ได้ถูกนำไปประยุกต์ใช้กับงานวิจัยต่อเนื่องอื่นที่ผู้วิจัยเป็นที่ปรึกษา อาทิ

1. รติกร มะกรครรภ์ “ระบบฐานข้อมูลงานวิจัย”, ปริญญานิพนธ์ สจพ. 2544 ได้รับ รางวัลชมเชย ประเภทนักศึกษา รางวัลนวัตกรรมเทคโนโลยีประจำปี 2544 โดยในงานนี้ได้พัฒนาเครื่องมือเพื่อทำการสืบค้นงานวิจัยจากข้อมูล 2 แหล่ง อาทิ จากฐานข้อมูลงานวิจัยเพื่อเก็บรวบรวมข้อมูลเกี่ยวกับงานวิจัยของสถาบันฯ ซึ่งพัฒนาขึ้นในงานปริญญานิพนธ์นี้

เพื่อให้เป็นแหล่งข้อมูลสำหรับฐานข้อมูลวิจัย และยังสืบค้นข้อมูลจากฐานข้อมูลผ่านเว็บของสำนักหอสมุดกลาง ของสถาบันฯ ซึ่งช่วยป้องกันและเป็นการลดปัญหาการรั่วไหลของข้อมูลหากสืบค้นผ่าน Backend หรือฐานข้อมูลสำนักหอสมุดกลางซึ่งใช้ระบบ Innopac โดยตรง

2. S. Sukaphat, B. Harangsri, Y. Temtanapat, "Refined Natural Language Questions with Context Analysis", The 8th World Multiconference on Systemics, Cybernetics and Information, Orlando, July 2004. ซึ่งเป็นบทความวิจัยที่ใช้แนวคิดการสืบค้นจากหลายแหล่ง แต่ปรับปรุงในส่วนของการทำ refinement ในข้อคำถามโดยอาศัยพื้นฐานในด้านความหมายของประโยค เช่น หากผู้ใช้ตั้งคำถามเป็น What is an ISA hard disk? ถ้าคำถามเช่นนี้ไปยัง Search Engine ทั่วไป คำหลายคำที่ปรากฏอยู่ในคำค้นจะเป็น Stop Words ซึ่งจะถูกตัดทิ้ง นอกจากนี้เนื่องจาก Search Engine สร้างคำค้นจากคำที่ปรากฏในเอกสาร ดังนั้นหากเปลี่ยนข้อคำถามจาก What is เป็นคำ definition ซึ่งให้ความหมายตามเจตนาของผู้ใช้ ก็อาจทำให้ค้นพบเอกสารที่ตรงตามความต้องการมากกว่า เนื่องจากใช้คำค้นที่อาจมีโอกาสปรากฏมากกว่าในเอกสาร อย่างไรก็ตามแต่ละผู้ใช้อาจมีความต้องการที่แตกต่างกันไป ดังนั้นการสร้าง MetaSearch โดยส่งคำค้นทั้งรูปแบบที่ผู้ใช้ถามจริง กับคำค้นที่มีความหมายเหมือนกันเพื่อค้นหาทั้งสองแนวทาง จะช่วยให้ผู้ใช้มีทางเลือกและมีโอกาสได้รับคำตอบที่ตรงใจกับการค้นหาได้มากขึ้น

แนวทางในการพัฒนาต่อ

เครื่องมือที่สร้างขึ้นนี้สามารถใช้ในการค้นคำจากหลายแหล่งโดยอาศัยแหล่งต้นตอต่าง ๆ สำหรับการพัฒนาต่อไปในอนาคต อาจปรับปรุงในเรื่อง

1 การจัดลำดับ (Ranking) โดยอาจให้น้ำหนักกับผลลัพธ์ที่มาจากทุกแหล่งมากกว่าแหล่งใดแหล่งหนึ่ง หรืออาจให้เรียงลำดับโดยให้น้ำหนักกับแหล่งใดแหล่งหนึ่งมากกว่าแหล่งอื่น ๆ ซึ่งอาจมีปัญหาในเรื่องความเร็วของการตอบกลับของ Search Engine เพราะลำดับที่ถูกเรียงไว้ก่อนแล้วอาจต้องเปลี่ยนแปลงไปเมื่อได้ผลลัพธ์กลับมาจากแหล่งอื่น ๆ เพิ่มขึ้น และซึ่งหากรอให้ได้คำตอบจากทุก Engine อาจจะให้ผลลัพธ์กับผู้ใช้ช้าจนเกินไป ดังนั้นจึงอาจต้องหาวิธีการที่ให้สมดุลที่ดีระหว่างลำดับความเหมือนและความเร็วในการตอบกลับ

2 การรองรับการขยายตัว ทั้งในเรื่องของจำนวนผู้ใช้และการเก็บ cache คำตอบจำนวนมากบนระบบจัดการฐานข้อมูล

3 การหาแหล่งข้อมูลต้นทางที่อาจให้ความหลากหลายในการตอบคำถาม เช่นแหล่งต้นทางในการค้นหาแผนที่ การค้นหาเฉพาะเรื่องทางด้านใดด้านหนึ่ง (เช่น สุขภาพ, โรงพยาบาล, แพทย์) เพื่อสร้างเป็นแหล่งบริการการค้นหาที่ให้ความหลากหลายในรูปแบบของคำตอบ

รายการเอกสารอ้างอิง

[1] 1Blink, http://www.1blink.com/, on April 1999

[2] All4One, http://www.all4one.com/, on April 1999

[3] AltaVista, http://www.altavista.com/, on April 2007

[4] Ask.com, http://www.ask.com/, on April 2007

[5] Argus Music Search, http://www.argus.com, on April 1999

[6] Forms in HTML documents, http://www.w3.org/TR/html4/interact/forms.html, on January 2006

[7] J. Friedl, *Mastering Regular Expressions*, 2nd Edition, O'Reilly & Associates Inc., 2002

[8] Google, http://www.google.com/, on April 2007

[9] Google SOAP Search API, http://code.google.com/apis/soapsearch/api_faq.html, on January 2006

[10] Highway61, http://www.highway61.com/, on April 1999

[11] Hotbot, http://www.hotbot.com/, on April 2007

[12] Information Retrieval - Wikipedia, http://en.wikipedia.org/wiki/Information_retrieval, on April 2007

[13] Isee, http://www.cs.rpi.edu/research/isee.html, on April 1999

[14] Lycos, http://www.lycos.com/, on April 2007

[15] WeiYi Meng, Clement Yu, King-Lup Liu, "Building Efficient and Effective Metasearch Engines", ACM Computing Surveys, Vol. 34, No. 1, March 2002, p 48-89

[16] MetaCrawler, http://metacrawler.cs.washington.edu:8080/, on April 1999

[17] Metasearch, http://metasearch.com/, on April 1999

[18] RFC 3986, http://rfc.net/rfc3986.html, on April 2007

[19] SavvySearch, http://www.savvysearch.com/, on April 1999

[20] A. Silberschatz, H. Korth, S. Sudarchan, *Database System Concepts*, 4th Ed., McGraw-Hill, 2002

[21] D.L. Stevens, D.E. Comer, *Internetworking With TCP/IP Volume III Client-Server Programming and Application BSD Socket Version,* Prentice-Hall, 1993

[22] W.R. Stevens, *Unix Network Programming*, Prentice Hall, 1990

[23] A. S. Tanenbaum, M. v. Steen, *Distributed Systems Principles and Paradigm*s, Prentice Hall, 2002

ภาคผนวก

ภาคผนวก 1: Profile ของการดึงข้อมูล

ภาคผนวก 2: โค้ดภาษาซีของเวอร์ชันเดิม

ภาคผนวก 1

Profile ของการดึงข้อมูล

1. AltaVista Profile

```
#top
#Pattern=\<br class='lb'>(.*?\</a> )
#attribute
#Pattern=\<a
    class='res'.href='.*?/\*\*http%3a//([^']*?)'>(.*?)\</a>(\<br>.*?\</s
    pan>)
#next
#Pattern=\<a href="([^"]*)".*">Next\s+&gt;&gt;\</a>
#found
#Pattern=\<DIV class=xs>.*?AltaVista\sfound\s([0-9,]*?)\sresult
```

2. Google Profile

```
#top
#Pattern=\<h2 class=r>(.*?\</table>)
#attribute
#Pattern=\<a href="([^"]*)"[^>]*>(.*?)\</a>\</h2>(.*?\</table>)
#next
#Pattern=\<a href=([^>]*)>\<div id=nn>\</div>Next\</a>
#found
#Pattern=Results.*?of\sabout\s\<b>([0-9,]*?)</b>.*?\<div id=res>
```

3. Lycos Profile

```
#top
#Pattern=\<ol class="rslts_Lst" .*?(<li>.*?)?\<div class="rslts_Nav">
#attribute
#Pattern=\<a href=".*?\&u=([^']+)';".*?>(.*?)\</a>(.*?)\</span>
#next
#Pattern=\<a\s+href="([^"]*)">Next</a> 
#found
#Pattern=<div class="rslts_Web">.*?of\s([0-9,]*?)\s\(.*?\<ol
class="rslts_Lst"
```

หมายเหตุ เนื่องจาก Lycos ใช้ tag <li> สำหรับกำกับทั้ง item ผลลัพธ์และรายละเอียดเช่นการโฆษณา ทำให้ไม่สามารถเลือกตัดเฉพาะแต่ item เท่านั้น ดังนั้นจึงต้องตัดทั้งกลุ่มของ result แล้วจึงมาเลือกตัดเฉพาะส่วนที่จะใช้อีกครั้งหนึ่งแทน

ภาคผนวก 2

โค้ดภาษาซีของเวอร์ชันเดิม

```
/*----- Server.c -----*/
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<stdio.h>
#include<signal.h>
#include"module.h"
#include<sys/time.h>
#include<unistd.h>

main()
{
   char c,word[50],str[1024],words[1024],str1[1024];
   FILE *fp;
   int fromlen,pid,PD,PD1,PD2,PD3,finx,add;
   char hostname[64],fname[10],*p,*q;
   register int s,ns;
   struct hostent *hp;
   struct sockaddr_in sin,fsin;
   fd_set  readfds;
   int     nfds;
   int pipel[2],pipea[2],pipeh[2],newfd;

   gethostname(hostname,sizeof(hostname));

   if((hp=gethostbyname(hostname))==NULL){
    fprintf(stderr,"%s: unknown host.\n",hostname);
    exit(1);
   }

   if((s=socket(AF_INET, SOCK_STREAM, 0))<0){
    perror("server: socket");
    exit(1);
   }

   sin.sin_family = AF_INET;
   sin.sin_port = htons(1234);
   bcopy(hp->h_addr, &sin.sin_addr, hp->h_length);

   if(bind(s,(struct sockaddr *)&sin,sizeof(sin))<0){
    perror("server: bind");
    exit(1);
   }

   if(listen(s,5)<0){
    perror("server: listen");
    exit(1);
   }

   pid=fork();
   if(pid!=0) exit(0);

   setsid();
   chdir("/var/lib/httpd/cgi-bin/search/");
```

```
   while((ns=accept(s,(struct sockaddr *)&fsin,&fromlen))>0){
    signal(SIGCHLD,SIG_IGN);
    pid=fork();

    if(pid==0){
       break;
    }
   }

    close(s);

    fp=fdopen(ns,"r");
           fgets(word,50,fp);
    fclose(fp);

    PD=getpid();

//   --------------- search

    pipe(pipel);
    pipe(pipea);
    pipe(pipeh);

    pid=fork();
    if(pid==0){
       close(pipel[0]);
       lycos(word,pipel[1]);
    }
    else
    {
       PD1=pid;
         pid=fork();
         if(pid==0)
                  altavista(word,pipea[1]);
         else
           {
        PD2=pid;
              pid=fork();
              if(pid==0)
                 hotbot(word,pipeh[1]);
              else
           PD3=pid;
      }
    }
     //  ------------   wait message

   if(pid!=0){
    close(pipel[1]);
    close(pipea[1]);
    close(pipeh[1]);

    if((fp=fopen("now","r"))==NULL){
       finx=1;
    }else{
       fscanf(fp,"%d",&finx);
       fclose(fp);
    }

    fp=fopen("now","w");
       if(finx<65535)
         fprintf(fp,"%d",finx+1);
```

```
   else
     fprintf(fp,"1");
fclose(fp);

sprintf(fname,"db/%d",finx);

fp=fopen("index","a+");
  fprintf(fp,"%s\t%s\t24\n",word,fname);
fclose(fp);

    while(1){
       FD_ZERO(&readfds);
       FD_SET(pipel[0], &readfds);
       FD_SET(pipea[0], &readfds);
       FD_SET(pipeh[0], &readfds);

   if(kill(PD1,0)!=0) FD_CLR(pipel[0],&readfds);
   if(kill(PD2,0)!=0) FD_CLR(pipea[0],&readfds);
   if(kill(PD2,0)!=0) FD_CLR(pipeh[0],&readfds);

   nfds=pipel[0];
   if(pipea[0]>nfds) nfds=pipea[0];
   if(pipeh[0]>nfds) nfds=pipeh[0];

       nfds = select(nfds+1, &readfds, NULL, NULL, NULL);
       if(nfds < 1){
          continue;
       }
       if(FD_ISSET(pipel[0], &readfds)){

      bzero(words,sizeof(words));

      read(pipel[0],words,sizeof(words));

      newfd=dup(pipel[0]);
      close(pipel[0]);
      pipel[0]=newfd;

      strcpy(str1,words);
      p=strstr(str1,"<a");
      q=strstr(p,">");q++;*q='\0';
      add=1;

      if((fp=fopen(fname,"r"))!=NULL){
  while(fgets(str,sizeof(str),fp)!=NULL){
     if(strstr(str,p)!=NULL){
        add=0;
        break;
     }
  }
      }else add=1;

      if(add==1){
        fp=fopen(fname,"a+");
        fputs(words,fp);
        fclose(fp);
      }

   }else
       if(FD_ISSET(pipea[0], &readfds)){
```

```
        bzero(words,sizeof(words));
        read(pipea[0],words,sizeof(words));

        newfd=dup(pipea[0]);
        close(pipea[0]);
        pipea[0]=newfd;

        strcpy(str1,words);
        p=strstr(str1,"<a");
        q=strstr(p,">");q++;*q='\0';
        add=1;

        if((fp=fopen(fname,"r"))!=NULL){
    while(fgets(str,sizeof(str),fp)!=NULL){
       if(strstr(str,p)!=NULL){
          add=0;
          break;
       }
    }
        }else add=1;

        if(add==1){
          fp=fopen(fname,"a+");
          fputs(words,fp);
          fclose(fp);
        }

     }else
          if(FD_ISSET(pipeh[0], &readfds)){

        bzero(words,sizeof(words));
        read(pipeh[0],words,sizeof(words));

        newfd=dup(pipeh[0]);
        close(pipeh[0]);
        pipeh[0]=newfd;

        strcpy(str1,words);
        p=strstr(str1,"<a");
        q=strstr(p,">");q++;*q='\0';
        add=1;

        if((fp=fopen(fname,"r"))!=NULL){
    while(fgets(str,sizeof(str),fp)!=NULL){
       if(strstr(str,p)!=NULL){
          add=0;
          break;
       }
    }
        }else add=1;

        if(add==1){
          fp=fopen(fname,"a+");
          fputs(words,fp);
          fclose(fp);
        }
     }else
     if((kill(PD1,0)!=0)&&(kill(PD2,0)!=0)&&(PD3,0)!=0)
    exit(0);
       }
}
```

```
   exit(0);
}


/*----- altavist.c -----*/
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<string.h>
#include<stdio.h>

altavista(char *Ps,int PPP)
{
  char str[256],*p,*q,*r;
  char ps[1024],ps1[1024];
  int s,i,newPPP;

  FILE *fp;

  char hostname[64];
  register int S;
  struct hostent *hp;
  struct sockaddr_in sin;

  if((s=call_socket("www.altavista.com")) < 0) {
    herror("Connection error");
    return 1;
  }

  sprintf(str,"GET http://www.altavista.com/cgi-
bin/query?pg=q&kl=XX&q=%s HTTP/1.0\n",Ps);
  writeln(s,str);

  while(1){

     bzero(ps,sizeof(ps));
     readln(s,ps);

     while(strstr(ps,"<dl><dt><b>")==NULL){
        bzero(ps,sizeof(ps));
        readln(s,ps);
    if(strstr(ps,"</HTML>")!=NULL)
      return;
     }

     while(strstr(ps,"Result Pages:")==NULL){
        if((p=strstr(ps,"<a"))==NULL) break;
        q=strstr(p,"<dd>");
        *q='\0';


            bzero(ps1,sizeof(ps1));
            readln(s,ps1);

        if((r=strstr(ps1,"<b"))==NULL) break;
        *r='\0';

        q=(char *)malloc((strlen(p)+strlen(ps1)));
        bzero(q,sizeof(q));
```

```
        sprintf(q,"A: %s<br>%s\n",p,ps1);

        write(PPP,q,strlen(q));

        free(q);

        newPPP=dup(PPP);
            close(PPP);
        PPP=newPPP;

            readln(s,ps);
            bzero(ps,sizeof(ps));
            readln(s,ps);
    }

    while(strstr(ps,"Next &gt;&gt;")==NULL){
           bzero(ps,sizeof(ps));
       readln(s,ps);
       if(strstr(ps,"</HTML>")!=NULL)
         return;
    }

    if((p=strstr(ps,"/cgi-bin/query?"))==NULL){
       return;
    }

    q=strstr(p,">");q--;
    *q='\0';
        sprintf(str,"GET http://www.altavista.com%s HTTP/1.0\n",p);

    close(s);

    if((s=call_socket("www.altavista.com")) < 0) {
      herror("Connection error");
      return 1;
    }

        writeln(s,str);

  }
}

/*----- lycos.c -----*/
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<string.h>
#include<stdio.h>

lycos(char *Ps,int PPP)
{
  char str[256],*q,*p,*r;
  char ps[1024],ps1[1024];
  int s,i,j,newPPP;

  FILE *fp;

  char hostname[64];
  register int S;
  struct hostent *hp;
```

```
  struct sockaddr_in sin;

  if((s=call_socket("www.lycos.com")) < 0) {
    herror("Connection error");
    return 1;
  }

  sprintf(str,"GET http://www.lycos.com/cgi-
bin/pursuit?matchmode=and&cat=lycos&query=%s HTTP/1.0\n",Ps);
  writeln(s,str);

  while(1){

    do{
        bzero(ps,sizeof(ps));
   readln(s,ps);
   if(strstr(ps,"</HTML>")!=NULL)
      return;
    }while(strstr(ps,"Matching Web Pages")==NULL);

   do{
          bzero(ps,sizeof(ps));
          readln(s,ps);
       }while(strstr(ps,"http://")==NULL);

     do{
        if((p=strstr(ps,"<a"))==NULL) break;
        if((q=strstr(p,"</b>"))==NULL) break;
        *q='\0';

            bzero(ps1,sizeof(ps1));
            readln(s,ps1);

            bzero(ps1,sizeof(ps1));
            readln(s,ps1);
            if((r=strstr(ps1,"<BR>"))==NULL) break;
            *r='\0';

        q=(char *)malloc(strlen(p)+strlen(ps1));
        sprintf(q,"L: %s%s\n",p,ps1);

        write(PPP,q,strlen(q));

        free(q);

        newPPP=dup(PPP);
        close(PPP);
        PPP=newPPP;

              readln(s,ps);
              readln(s,ps);
              readln(s,ps);
              readln(s,ps);
              readln(s,ps);
              bzero(ps,sizeof(ps));
              readln(s,ps);
     }while(strlen(ps)>5);

           while(strstr(ps,"next &gt;&gt;")==NULL){
             bzero(ps,sizeof(ps));
             readln(s,ps);
```

```
        if(strstr(ps,"</HTML>")!=NULL)
           return;
      }

      if((p=strstr(ps,"/cgi-bin/pursuit?"))==NULL){
        return;
      }
      q=strstr(p,">");q--;
      *q='\0';
          sprintf(str,"GET http://www.lycos.com%s HTTP/1.0\n",p);

          close(s);

          if((s=call_socket("www.lycos.com")) < 0) {
            herror("Connection error");
            return 1;
          }

          writeln(s,str);

  }
}

/*----- hotbot.c -----*/
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<string.h>
#include<stdio.h>

hotbot(char *Ps,int PPP)
{
  char str[256],*q,*p,*r,*a,*b;
  char ps[65536],ps1[1024],ps2[1024];
  int s,i,j,FIRST=0,newPPP;

  FILE *fp;

  char hostname[64];
  register int S;
  struct hostent *hp;
  struct sockaddr_in sin;

  if((s=call_socket("www.hotbot.com")) < 0) {
    herror("Connection error");
    return 1;
  }

  sprintf(str,"GET http://www.hotbot.com/default.asp?MT=%s
HTTP/1.1\n",Ps);
  writeln(s,str);

  while(1){
     while(strstr(ps,"/director.asp?target=")==NULL){
        bzero(ps,sizeof(ps));
    readln(s,ps);
    if(strstr(ps,"</html>")!=NULL)
       return;
     }
```

```
  while(strlen(ps)>8){
   if(FIRST==0){
     if((p=strstr(ps,"=DH"))==NULL) break;
     p+=5;
     q=strstr(p,"</b>");
     *q='\0';

         bzero(ps1,sizeof(ps1));
         readln(s,ps1);
     if((q=strstr(ps1,"<br>"))==NULL) break;
     *q='\n';q++;*q='\0';

         bzero(ps2,sizeof(ps2));
         readln(s,ps2);
     if((a=strstr(ps2,"http://"))==NULL) break;
     q=strstr(a,"<BR>");
     *q='\0';
     sprintf(ps2,"<a href=%s>",a);
   }else
       {
    if(FIRST==1){
      if((p=strstr(ps,"INK_DH"))==NULL) break;
    }else p=b;

      p+=7;
      q=strstr(p,"</b>");
      r=q+8;
      *q='\0';
      if((q=strstr(r,"<br>"))==NULL) break;
      if((a=strstr(q,"http://"))==NULL) break;
      *q='\0';
      strcpy(ps1,r);
      if((q=strstr(a,"<BR>"))==NULL) break;
      if((b=strstr(q,"INK_DH"))==NULL) break;
      *q='\0';
      strcpy(ps2,a);
      FIRST++;
        }

     r=(char *)malloc(strlen(p)+strlen(ps1)+strlen(ps2));
     sprintf(r,"H: %s%s<br>%s",ps2,p,ps1);

     newPPP=dup(PPP);
     close(PPP);
     PPP=newPPP;

     write(PPP,r,strlen(r));

     free(r);

  if(FIRST==0){
         readln(s,ps);
         bzero(ps,sizeof(ps));
         readln(s,ps);
  }

  if(FIRST>10) break;

    }  //  While

FIRST=0;
```

```
            while(strstr(ps,"?MT=")==NULL){
              bzero(ps,sizeof(ps));
              readln(s,ps);
          if(strstr(ps,"</html>")!=NULL)
             return;
       }

       q=strstr(p,">");q--;
       *q='\0';

            sprintf(str,"GET http://www.hotbot.com/default.asp%s
HTTP/1.1\n",p);

       if(FIRST==0)  FIRST=1;
            close(s);

            if((s=call_socket("www.hotbot.com")) < 0) {
              herror("Connection error");
              return 1;
            }

            writeln(s,str);
  }
}

/*----- client.c -----*/
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

extern int errno;

main()
{
   char c,*msg,*ENV,*p,*q;
   FILE *fp;
   char hostname[64],er1[1024],er2[256],buf[1024];
   register int s;
   struct hostent *hp;
   struct sockaddr_in sin;
   int er3,found=0;

   ENV=getenv("CONTENT_LENGTH");
   msg=malloc(atoi(ENV));
   gets(msg);

   p=strstr(msg,"=");
   msg=q=++p;

   while(*p!='\0'){
      if(*p=='%'){
    p++;
    sscanf(p,"%2x",q);
    p+=2;
    q++;
      }
      else{
```

```
    *(q++)=*(p++);
      }
   }

   puts("Content-type: text/html\n\n");

   if((fp=fopen("index","r"))!=NULL){
      while(fgets(buf,1024,fp)!=NULL){
    sscanf(buf,"%s%s%d",er1,er2,&er3);
    if(strcmp(er1,msg)==0){
       found=1;
       break;
    }
      }
   }

   if(found==1){
     printf("<html><head><META http-equiv=refresh content=\"0; url=/cgi-
bin/search/watch.cgi?word=%s\"></head>",msg);
     printf("<h1>Searching....  %s</h1>\n",msg);
     printf("<center>Wait a minute...... <center>");
     exit(0);
   }

   gethostname(hostname,sizeof(hostname));

   if((hp=gethostbyname(hostname))==NULL){
    printf("%s: unknown host.\n",hostname);
    exit(1);
   }

   if((s=socket(AF_INET, SOCK_STREAM, 0))<0){
    printf("client: socket");
    exit(1);
   }

   sin.sin_family = AF_INET;
   sin.sin_port = htons(1234);
   bcopy(hp->h_addr, &sin.sin_addr, hp->h_length);

   if(connect(s,(struct sockaddr *)&sin,sizeof(sin))<0){
    printf("client: connect");
    exit(1);
   }

   send(s,msg,strlen(msg),0);

//--------------------------------------<begin search>

   printf("<html><head><META http-equiv=refresh content=\"20; url=/cgi-
bin/search/watch.cgi?word=%s\">",msg);
   puts("<title>Waitting.. for search</title></head>");
   puts("<body bgcolor=ffffff><center><img
src=/~yod/images/www.gif></center><br>");
   printf("<h1>Searching....  %s</h1>\n",msg);
   printf("<center>Wait 20 seconds...... <center></body></html>");

//--------------------------------------<End>

   close(s);
   exit(0);
```

```
}

/*----- eraser.c -----*/
#include<stdio.h>
#include<string.h>

main()
{
   FILE *fp,*fq;
   int pid,Time;
   char buf[1024],p[1024],q[256];

   pid=fork();
   if(pid!=0)
    exit(0);

   setsid();

   chdir("/var/lib/httpd/cgi-bin/search/");

   while(1){
    if((fp=fopen("index","r"))!=NULL){
       fq=fopen("recheck","w");
       while(fgets(buf,1024,fp)!=NULL){
      sscanf(buf,"%s%s%d",p,q,&Time);
      if(Time>1)
        fprintf(fq,"%s\t%s\t%d\n",p,q,Time-1);
      else
        unlink(q);
       }
       fclose(fp);
       fclose(fq);
       fp=fopen("index","w");
       fq=fopen("recheck","r");
      while(fgets(buf,1024,fq)!=NULL)
        fputs(buf,fp);
       fclose(fp);
       fclose(fq);
    }
    sleep(3600);
   }
}


/*----- module.h -----*/
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<string.h>
#include<stdio.h>

int call_socket(char *hostname)
{
   struct sockaddr_in sa;
   struct hostent     *hp;
   int    a,ss;

   if((hp=gethostbyname(hostname))==NULL) return(-1);

   bzero(&sa, sizeof(sa));
```

```
   bcopy(hp->h_addr, (char *)&sa.sin_addr, hp->h_length);
   sa.sin_family = hp->h_addrtype;
   sa.sin_port = htons((u_short)80);

   if((ss=socket(hp->h_addrtype, SOCK_STREAM, 0)) < 0)
      return(-1);
   if(connect(ss,(struct sockaddr *)&sa, sizeof(sa)) < 0) {
      close(ss);
      return(-1);
   }
   return(ss);
}

void writeln(int ss,char *buf)
{
  write(ss, buf, strlen(buf));
  write(ss, "\n", 1);
}

int readln(int ss,char *buf)
{
  int to=0;
  char c;

  do {
    read(ss, &c, 1);
    buf[to++] = c;
  } while (c != '\n');

  buf[to] = '\0';
}

/*----- watch.c -----*/
#include<stdio.h>
#include<string.h>

main()
{
    char
*buf,*msg,BUF[2000],*p,*q,er1[1024],er2[256],end[256]="Searching...";
    FILE *fp;
    int i,form,to,er3;

    buf=(char *)getenv("REQUEST_URI");
    msg=strstr(buf,"?word=");
    msg+=6;
    if((p=strstr(msg,"&b="))!=NULL){
      q=strstr(p,"=");
      *p='\0';
      q++;
      form=atoi(q);
      p=strstr(q,"&t=");
      q=strstr(p,"=");
      q++;
      to=atoi(q);
    }else{
      form=1;
      to=10;
     }

    puts("Content-type: text/html\n\n");
```

```
    puts("<HTML><Head><Title>Search Result</title></head>");
    puts("<Body bgcolor=ffffff>");
    puts("<img src=/images/www.gif><br>");
    puts("<table border=0 width=90%><tr><td bgcolor=5588ee>");
    printf("<font size=+2><b>Search Results</b></font>
for %s</td></tr></table><br>",msg);
/*
    if(form>1){
      puts("<center><table bgcolor=000000 border=0 cellpadding=0
cellspacing=0 align=left><tr><td>");
      puts("<table border=0 cellpadding=5 cellspacing=1><tr
bgcolor=ffffee><td>");
      printf("<a href=/cgi-bin/search/watch.cgi?word=%s&b=%d&t=%d><<
Previous page</a><br>",msg,form-10,to-10);
      puts("</td><tr></table></td></tr></table>");
    }else puts("<center>");

    puts("<table bgcolor=000000 border=0 cellpadding=0 cellspacing=0
align=right><tr><td>");
    puts("<table border=0 cellpadding=5 cellspacing=1><tr
bgcolor=ffffee><td>");
    printf("<a href=/cgi-bin/search/watch.cgi?word=%s&b=%d&t=%d>Next
page >></a><br>",msg,to+1,to+10);
    puts("</td><tr></table></td></tr></table></center><br><br>");
*/
    fp=fopen("index","r");
    while(fgets(BUF,2000,fp)!=NULL){
       sscanf(BUF,"%s%s%d",er1,er2,&er3);
       if(strcmp(er1,msg)==0)
      break;
    }
    fclose(fp);

       fp=fopen(er2,"r");
       for(i=0;i<((form-1));i++)
          fgets(BUF,2000,fp);

        for(i=0;i<10;i++){
          if(fgets(BUF,2000,fp)==NULL){
         sprintf(end,"End.");
         break;
     }
          puts(BUF);puts("<br>");
    }

    close(fp);

     printf("<br><br><center>Status : <font
color=ff0000>%s</font></center><br>",end);

    if(form>1){
      puts("<br><center><table bgcolor=000000 border=0 cellpadding=0
cellspacing=0 align=left><tr><td>");
      puts("<table border=0 cellpadding=5 cellspacing=1><tr
bgcolor=ffffee><td>");
      printf("<a href=/cgi-bin/search/watch.cgi?word=%s&b=%d&t=%d><<
Previous page</a><br>",msg,form-10,to-10);
      puts("</td><tr></table></td></tr></table>");
    }else puts("<br><center>");
```

```
    if(strcmp(end,"End.")!=0){
       puts("<table bgcolor=000000 border=0 cellpadding=0 cellspacing=0
align=right><tr><td>");
       puts("<table border=0 cellpadding=5 cellspacing=1><tr
bgcolor=ffffee><td>");
       printf("<a href=/cgi-bin/search/watch.cgi?word=%s&b=%d&t=%d>Next
page >></a><br>",msg,to+1,to+10);
       puts("</td><tr></table></td></tr></table></center>");
       puts("</HTML>");
    }else{
       puts("</HTML>");
     }
}
```

ประวัติผู้วิจัย

ชื่อ นส. เยาวดี เต็มธนาภัทร์ (Yaowadee Temtanapat)

คุณวุฒิ

- **Ph. D , Computer Science,** December 1998, *Rensselaer Polytechnic Institute*, Troy, NY
- **Master of Science, Applied Statistics (Honors),** December 1989, *National Institute of Development Administration*
- **Bachelor of Science, Physiotherapy,** March 1987, *Mahidol University*

สาขาวิชาที่เชี่ยวชาญ Database Systems, Database System Security, Distributed Data Integration

ผลงานวิจัยที่พิมพ์ออกเผยแพร่

1. S. Adali, C. Bufi, **Y. Temtanapat**, "Integrated Search Engine", *1997 IEEE Knowledge and Data Engineering Exchange workshop*, Nov 1997

2. **Y. Temtanapat**, D. Spooner, "Detection of Access Control Flaws in A Distributed Database System with Local Site Autonomy", *1997 International Database Engineering and Application Symposium*, Aug 1997.

ที่ติดต่อ ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์
โทรศัพท์ภายใน 4626