เปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล

I/O INDEPENDENT EXPERT SYSTEM SHELL

นายสุกิจ เมฆจำเริญ

MR. SUKIT MEKJAMROEN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2540

ISBN 974-621-930-8

I/O INDEPENDENT EXPERT SYSTEM SHELL

SUKIT MEKJAMROEN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT' S INSTITUTE OF TECHNOLOGY LADKRABANG

1997

ISBN 974-621-930-8

หัวข้อวิทยานิพนธ์	เปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล
นักศึกษา	นายสุกิจ เมฆจำเริญ
อาจารย์ผู้คุมวิทยานิพนธ์	ศ.ดร.ศรีศักดิ์ จามรมาน
ระดับการศึกษา	วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า
ภาควิชา	วิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง
พ.ศ.	2540

บทคัดย่อ

การพัฒนาระบบผู้เชี่ยวชาญในปัจจุบัน วิธีที่ง่ายและรวดเร็ววิธีหนึ่งคือการใช้เปลือกระบบผู้เชี่ยวชาญ เปลือกระบบผู้เชี่ยวชาญในปัจจุบันมีหลายรูปแบบ ซึ่งโดยทั่วไปส่วนที่ทำหน้าที่ในการรับส่งข้อมูลจะเป็นส่วนหนึ่งของตัวเปลือกระบบผู้เชี่ยวชาญนั้นๆ ทำให้ผู้ใช้ไม่สามารถจะเปลี่ยนแปลงได้ โดยที่ระบบผู้เชี่ยวชาญที่สร้างขึ้นจากเปลือกระบบผู้เชี่ยวชาญแบบที่กล่าวถึงนี้จะสามารถเชื่อมต่อกับอุปกรณ์รับส่งข้อมูลได้เฉพาะอุปกรณ์ที่เปลือกระบบผู้เชี่ยวชาญนั้นกำหนดให้ วิทยานิพนธ์นี้ เป็นการวิจัยและพัฒนาเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูลโดยใช้ภาษา C บนระบบปฏิบัติการ UNIX ในการพัฒนา เปลือกระบบผู้เชี่ยวชาญระบบนี้โปรแกรมส่วนรับข้อมูล โปรแกรมการจัดการผลลัพธ์ และส่วนอนุมานจะเป็นส่วนที่แยกออกจากกัน

ในส่วนของโปรแกรมการรับข้อมูล และโปรแกรมการจัดการผลลัพธ์ ผู้ใช้สามารถเขียนโปรแกรมขึ้นเองได้ซึ่งเปลือกระบบผู้เชี่ยวชาญระบบนี้จะช่วยให้การสร้างระบบผู้เชี่ยวชาญสามารถทำได้อย่างเหมาะสมกับความต้องการของผู้ออกแบบระบบ และสามารถนำระบบผู้เชี่ยวชาญไปใช้งานได้หลายรูปแบบยิ่งขึ้น

Thesis Title	I/O Independent Expert System Shell
Student	Mr. Sukit Mekjamroen
Thesis Advisor	Prof. Dr. Srisakdi Chamonman
Level of Study	Master of Engineering in Electrical Engineer
Department	Computer Engineering Faculty of Engineering
	King Monkut's Institute of Technology Ladkrabang
Year	1997

ABSTRACT

One of the most easiest and fastest method to develop expert systems is using expert system shell. Normally, these expert system shells have the I/O (input/output) routines embedded in their program then the user cannot modify it. Moreover, the expert systems that build from these expert system shells can use only I/O devices which these expert system shells allow. This thesis is the research and development of I/O independent expert system shell by using C language on UNIX operating system and this expert system shell have the independent modules that control inputs, outputs, knowledge base and inference engine. For the users, they can write their own module to interface between their own devices and I/O independent expert system shell. This shell will help the expert system developers to build their systems in appropriate way.

กิตติกรรมประกาศ

วิทยานิพนธ์เรื่องเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูลฉบับนี้สำเร็จลงด้วยดีก็เพราะได้รับกรุณาความเอาใจใส่และคำแนะนำที่มีประโยชน์ จาก ศาสตราจารย์ ดร.ศรีศักดิ์ จามรมาน ผู้วิจัยขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ มูลนิธิเพื่อการสื่อสารและคอมพิวเตอร์ ที่ได้มอบทุนอุดหนุนการวิจัยจนวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงด้วยดี

ขอขอบพระคุณ คุณพรชัย วัฒนาพรที่ได้ให้โอกาสให้ผู้วิจัยได้ใช้เวลาสถานที่และระบบคอมพิวเตอร์ในการทำงาน

สุดท้ายนี้ผู้วิจัยขอขอบพระคุณ คุณอรัญญา ใคเสียง และ คุณเอกวีร์ รุจิพัฒนพงศ์ ผู้ที่ให้ความช่วยเหลือและข้อเสนอแนะที่ดีแก่ผู้วิจัย

สุกิจ เมฆจำเริญ

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญภาพ

หน้า

สารบัญภาพ (ต่อ)

หน้า

บทที่ 1
บทนำ

ระบบผู้เชี่ยวชาญเป็นสาขาย่อยสาขาหนึ่งของปัญญาประดิษฐ์ ซึ่งใช้ช่วยในการสรุปหาคำตอบหรือแก้ไขปัญหาเฉพาะด้าน ซึ่งต้องอาศัยความรู้หรือประสบการณ์ในด้านนั้นๆ สำหรับการแก้ปัญหาซึ่งในอดีตต้องอาศัยผู้ที่มีความเชี่ยวชาญเป็นพิเศษ แต่ในปัจจุบันเราสามารถนำความรู้ที่ได้จากประสบการณ์ของผู้เชี่ยวชาญ หรือจากตำรามาเก็บไว้ในโปรแกรมคอมพิวเตอร์ ที่เรียกว่าระบบผู้เชี่ยวชาญ ซึ่งโปรแกรมระบบผู้เชี่ยวชาญนี้จะแปลงรูปของความรู้ที่ได้ให้อยู่ในรูปที่ง่ายต่อการประมวลผลเพื่อนำไปเก็บในฐานความรู้ (Knowledge Base) เมื่อผู้ใช้ต้องการทราบข้อสรุปหรือคำตอบในเรื่องที่เกี่ยวข้องกับความรู้ในระบบผู้เชี่ยวชาญ โปรแกรมระบบผู้เชี่ยวชาญจะนำความรู้ที่เก็บไว้นี้มาใช้ในการประมวลผลโดยจะมีส่วนอนุมาน(Inference Engine) ทำหน้าที่หาคำตอบหรือข้อสรุปที่ได้ส่งให้กับผู้ใช้ ความเชื่อถือได้ของระบบผู้เชี่ยวชาญจะขึ้นอยู่กับความถูกต้องและปริมาณความรู้ที่ใส่เข้าไปในระบบ ระบบที่มีความรู้เก็บไว้มากก็จะสามารถวิเคราะห์ปัญหาได้อย่างแม่นยำใกล้เคียงกับการวิเคราะห์ของผู้เชี่ยวชาญมากขึ้น

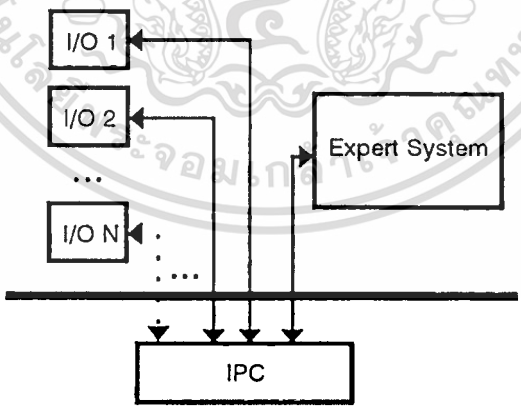
ความเป็นมาและประโยชน์ที่จะได้รับจากงานวิจัย

ในการพัฒนาระบบผู้เชี่ยวชาญทางด้านต่างๆนั้นสามารถทำได้โดยการสร้างโปรแกรมเพื่อพัฒนาระบบนั้นขึ้นมาแล้วใส่ความรู้ให้กับระบบ แต่ในการสร้างระบบโดยเริ่มจากการเขียนโปรแกรมนั้นจะทำได้ช้ายุ่งยากและซับซ้อน ในปัจจุบันได้มีการสร้างตัวโปรแกรมระบบผู้เชี่ยวชาญสำเร็จรูปที่มีเฉพาะเปลือกข้างนอกแต่ไม่มีข้อมูลความรู้ซึ่งโปรแกรมนี้เรียกกันว่า เปลือกระบบผู้เชี่ยวชาญ (Expert System Shell) ในการสร้างระบบผู้เชี่ยวชาญโดยใช้เปลือกระบบผู้เชี่ยวชาญนี้สามารถทำได้โดยง่ายไม่ต้องอาศัยความรู้ในเรื่องการเขียนโปรแกรมมากนัก ผู้พัฒนาระบบผู้เชี่ยวชาญเพียงแต่ใส่ความรู้ลงไปในเปลือกระบบผู้เชี่ยวชาญเท่านั้น ทำให้การสร้างระบบผู้เชี่ยวชาญสามารถทำได้ในเวลารวดเร็ว และง่ายขึ้น

แต่ในบางครั้งผู้พัฒนาระบบผู้เชี่ยวชาญพบว่า ในการพัฒนาระบบนั้นบางครั้งผู้พัฒนาไม่สามารถใช้เปลือกระบบผู้เชี่ยวชาญในการพัฒนาได้ เพราะเปลือกระบบผู้เชี่ยวชาญที่ต้องการไม่สามารถที่จะใช้งานร่วมกับอุปกรณ์รับส่งข้อมูล (input/output) ที่ผู้พัฒนาระบบต้องการ ถึงแม้ว่าเปลือกระบบผู้เชี่ยวชาญนั้นจะมีส่วนอนุมานความรู้ ซึ่งเป็นส่วนสำคัญที่สุดของระบบ ที่ทำงานได้ตามที่ผู้พัฒนาระบบต้องการก็ตาม

จากปัญหานี้คือที่มาของการวิจัยในการพัฒนา เปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล โดยในระบบนี้ส่วนรับส่งข้อมูลไม่ได้ฝังตัวอยู่ในระบบ แต่จะถูกแยกออกโดยจะติดต่อกับตัวระบบโดยผ่านทางช่องทางการติดต่อระหว่างโปรเซส (IPC: InterProcess Communication) ซึ่งโปรแกรมที่จะทำหน้าที่เป็นส่วนรับส่งข้อมูลจะเป็นโปรแกรมใดก็ได้ ที่สามารถส่งและรับข้อมูลในรูปแบบที่กำหนดกับส่วนติดต่อระหว่างโปรเซสได้ดังรูปที่ 1.

ปัญหานี้น่าสนใจเพราะ ถ้าเราสามารถพัฒนาเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูลได้ เปลือกระบบผู้เชี่ยวชาญแบบนี้จะสามารถใช้งานได้อย่างมีประสิทธิภาพมากขึ้น เพราะถ้าต้องการใช้อุปกรณ์รับส่งข้อมูลชนิดอื่น ก็สามารถทำได้โดยเขียนเฉพาะส่วนรับส่งข้อมูลเพิ่มเท่านั้น ไม่ต้องเขียนใหม่ทั้งระบบ ซึ่งเป็นการประหยัดเวลาและทรัพยากรอย่างมาก



รูปที่ 1. แสดงแนวคิดพื้นฐานของเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล

วัตถุประสงค์และขอบเขตของการวิจัย

ในการวิจัยนี้มีวัตถุประสงค์ที่จะพัฒนาเปลือกระบบผู้เชี่ยวชาญอย่างง่ายที่ไม่ซับซ้อนและสามารถทำงานได้โดยไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล ส่วนประกอบต่างๆที่มีการใช้ในระบบ เช่น ส่วนอนุมาน ส่วนฐานความรู้ และส่วนอื่นๆ จะใช้วิธีที่ง่ายที่สุดในการพัฒนาและการจัดการ

โปรแกรมที่ได้จัดทำขึ้นจะเป็นลักษณะของ Client - Server โดยที่จะมีโปรแกรมที่คอยรับข้อมูลเพื่อนำไปอนุมานทำงานอยู่เป็นโปรเซสที่ทำงานอยู่เบื้องหลังเพื่อรอรับข้อมูลจากโปรแกรมที่จะมาขอใช้บริการซึ่งโปรแกรมเหล่านี้เป็นโปรแกรมที่ผู้ใช้เขียนขึ้นเอง โดยในโครงงานนี้จะสร้างโปรแกรมที่ใช้ติดต่อกับโปรแกรมส่วนอนุมานโดยผ่านทางจอภาพและผ่านทาง TCP/IP มาให้ด้วยสองโปรแกรม สำหรับผู้ใช้ที่ไม่ต้องการจะใช้อุปกรณ์รับส่งข้อมูลอื่นๆก็จะสามารถใช้โปรแกรมที่ติดต่อกับส่วนอนุมานทางจอภาพสำหรับสร้างระบบผู้เชี่ยวชาญเหมือนกับการใช้เปลือกระบบผู้เชี่ยวชาญปกติได้ และสำหรับผู้ใช้ที่ต้องการใช้งานอุปกรณ์อื่นผู้ใช้สามารถเขียนขึ้นเองได้โดยง่ายและมีตัวอย่างในเนื้อหาของวิทยานิพนธ์นี้

ตัวโปรแกรมจะพัฒนาโดยเป็นโปรแกรมเป็นภาษาซีบนระบบปฏิบัติการยูนิกซ์ซึ่งเครื่องที่ใช้ในการพัฒนาคือเครื่อง SUN Sparc 1+ และเครื่องมินิคอมพิวเตอร์ Stratus R/25 โดยที่ระบบปฏิบัติการบนเครื่องทั้งสองเป็นยูนิกซ์ SVR4 และเครื่องทั้งสองเชื่อมต่อกันโดยใช้โปรโตคอล TCP/IP บน Ethernet

เนื้อหาของวิทยานิพนธ์

วิทยานิพนธ์นี้แบ่งเนื้อหาออกเป็น 8 บทซึ่งแต่ละบทมีรายละเอียดดังนี้

บทที่ 1. เป็นบทนำ กล่าวถึงความเป็นมา ความสำคัญ วัตถุประสงค์ และ เนื้อหาโดยรวมของวิทยานิพนธ์นี้

บทที่ 2. กล่าวถึงความรู้เบื้องต้นในเรื่องของระบบผู้เชี่ยวชาญว่า ระบบผู้เชี่ยวชาญคืออะไร ส่วนประกอบมีอะไรบ้าง

บทที่ 3. อธิบายหลักการและส่วนประกอบของเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล

บทที่ 4. เป็นการอธิบายในเรื่องของ XAdmin ซึ่งเป็นส่วนประกอบของระบบสำหรับทำหน้าที่ในการควบคุมการทำงานของระบบและจัดการเกี่ยวกับการเก็บและรับความรู้จากผู้เชี่ยวชาญ

บทที่ 5. เป็นการอธิบายเรื่องของ XServer ซึ่งเป็นส่วนที่เป็นเปลือกระบบผู้เชี่ยวชาญทำหน้าที่ในการรับข้อมูลจาก IPC แล้วนำมาอนุมานจากนั้นก็จะนำผลลัพธ์ที่ได้ส่งต่อให้กับ IPC เพื่อส่งให้กับโปรแกรมที่ทำหน้าที่ควบคุมการทำงานของ อุปกรณ์ output

บทที่ 6. กล่าวถึงส่วน XClient ซึ่งเป็นส่วนของโปรแกรมที่ทำหน้าที่ในการรับส่งข้อมูล

บทที่ 7. เป็นการใช้งานโปรแกรมเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล

บทที่ 8. กล่าวถึงบทสรุปและข้อเสนอแนะ

ซึ่งหลังจากเนื้อหาทั้ง 8 บทแล้วก็จะเป็นภาคผนวกต่างซึ่งรวบรวมข้อมูลที่เป็นประโยชน์ต่อผู้อ่านคือ ภาคผนวก ก. รวบรวม Flow Chart ของ Module ต่างๆ, ภาคผนวก ข. เป็นการรวบรวม Source Code, ภาคผนวก ค. เป็นตัวอย่างโปรแกรมสำหรับอุปกรณ์รับส่งข้อมูล และท้ายสุดเป็นบรรณานุกรมซึ่งรวบรวมรายชื่อหนังสืออ้างอิงที่ใช้ในวิทยานิพนธ์นี้

บทที่ 2
ระบบผู้เชี่ยวชาญ

ระบบผู้เชี่ยวชาญคืออะไร และประวัติความเป็นมาของระบบผู้เชี่ยวชาญ

ระบบผู้เชี่ยวชาญคือโปรแกรมคอมพิวเตอร์ที่เก็บทั้งความรู้เกี่ยวกับปัญหาที่จะแก้และขบวนการอนุมานเพื่อนำไปสู่ผลสรุปหรือคำตอบของปัญหานั้น (วิลาศ วูวงศ์, บุญเจริญ ศิริเนาวกุล 2535 : 7) ความรู้ที่เก็บมีทั้งความรู้ที่เป็นความจริงที่อาจจะถูกบันทึกไว้ในรูปของตำราหรือเอกสารทางวิชาการและความรู้ที่ได้จากประสบการณ์ที่อาจจะไม่อยู่ในรูปของตำราหรือเอกสารทางวิชาการ แต่จะต้องดึงออกมาจากผู้เชี่ยวชาญหรือผู้ชำนาญที่มีประสบการณ์นั้น

การค้นคว้าในสาขาปัญญาประดิษฐ์เริ่มต้นหลังจากการประชุมของ McCarthy, Minsky, Simon และบุคคลอื่นๆ ในปี ค.ศ. 1956 (วิลาศ วูวงศ์, บุญเจริญ ศิริเนาวกุล 2535 : 8) การประชุมดังกล่าวมีวัตถุประสงค์เพื่อแลกเปลี่ยนความเห็นเกี่ยวกับการค้นคว้าทางด้าน "ความรู้" โดยใช้คอมพิวเตอร์ ในสมัยเริ่มแรกนี้มีแนวความคิดการหาทางทำให้คอมพิวเตอร์แก้ปัญหาปริศนาหรือเกม จะเป็นทางนำไปสู่การเข้าใจสิ่งที่เรียกกันว่า "ปัญญา" หรือ "ความรู้" ที่มนุษย์เรามีอยู่

เมื่อย่างเข้าทศวรรษที่ 1960 ได้มีการพัฒนาเทคนิคการค้นหาคำตอบ (search technique) ต่างๆ ที่เป็นผลพลอยได้จากการค้นคว้าเรื่องการแก้ปัญหาปริศนาและเกม จากนั้นก็มีการค้นคว้ากลไกในการแก้ปัญหาที่สามารถใช้ได้กับปัญหาทั่วไป ในปี 1960 McCarthy ได้พัฒนาภาษา LIPS ซึ่งเป็นภาษาหนึ่งที่ได้รับความสนใจมากที่สุดในการเขียนโปรแกรมเกี่ยวกับปัญญาประดิษฐ์

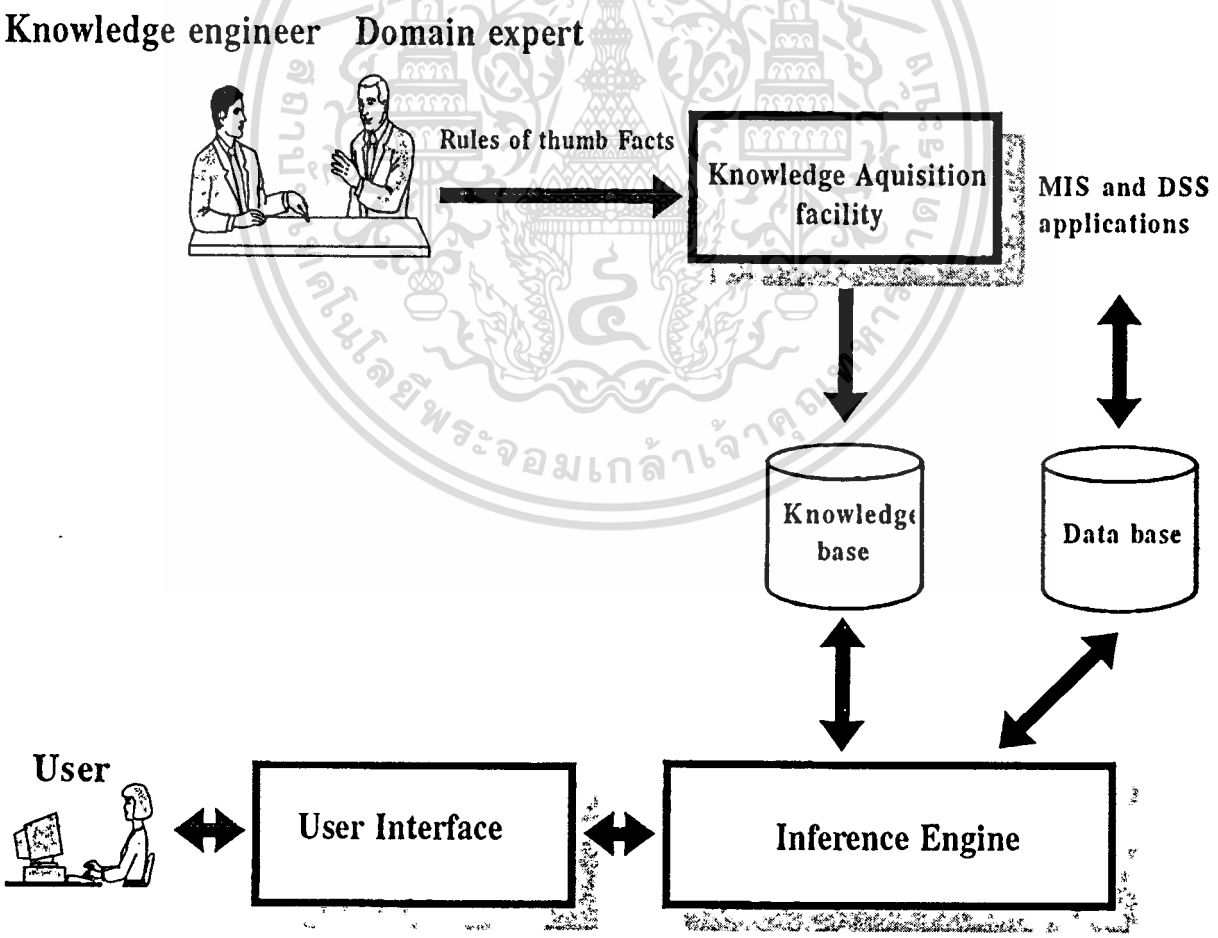
ครึ่งหลังของทศวรรษที่ 1960 ได้เริ่มมีการค้นคว้าพื้นฐานเกี่ยวกับการแสดงความรู้โดยที่ Newell ได้เสนอรูปแบบการแสดงความรู้ในรูปของระบบกฎ (Production System) ในปี ค.ศ. 1967 และ Quillian ได้เสนอรูปแบบข่ายความหมาย ในปี ค.ศ. 1968 และในปี ค.ศ. 1973 Colmerauer ได้เสนอภาษาใหม่เพื่อใช้ในการค้นคว้าปัญญาประดิษฐ์ขึ้นมาอีกภาษาหนึ่ง ชื่อ Prolog ซึ่งมีฐานอยู่บนวิชาตรรกวิทยา ภาษานี้กำลังได้รับความนิยมเพิ่มขึ้นในวงการปัญญาประดิษฐ์ในปัจจุบัน ในปี ค.ศ. 1974 Minsky ได้เสนอรูปแบบการแสดงความรู้ใหม่ชื่อกรอบ (frame) ขึ้น

เมื่อเข้าครึ่งหลังของทศวรรษที่ 1970 ได้เริ่มมีการสร้างระบบผู้เชี่ยวชาญขึ้น นับว่าเป็นการเริ่มประยุกต์ใช้วิชาปัญญาประดิษฐ์ ผลจากการนี้ทำให้ผู้คนเริ่มแบ่งแยกวิชาปัญญาประดิษฐ์ออกเป็น 2 แขนงคือแขนงที่เน้นการประยุกต์ ซึ่งก็คือวิศวกรรมความรู้ และแขนงที่เน้นด้านทฤษฎี

ในหัวข้อถัดไปจะเป็นการอธิบายถึงส่วนประกอบต่างๆของระบบผู้เชี่ยวชาญว่าระบบผู้เชี่ยวชาญทั่วๆไปมีส่วนประกอบสำคัญอะไรบ้าง

ส่วนประกอบของระบบผู้เชี่ยวชาญ

ระบบผู้เชี่ยวชาญโดยทั่ว ๆ ไป จะประกอบด้วยส่วนประกอบพื้นฐาน 5 ส่วน ดังแสดงใน **รูปที่** 2 ซึ่งในระบบผู้เชี่ยวชาญบางระบบอาจจะไม่มีส่วนประกอบครบทั้ง 5 ส่วนดังกล่าวข้างต้นก็ได้ แต่จะต้องมีฐานความรู้ (Knowledge base) และเครื่องอนุมาน (inference engine) เป็นอย่างน้อย



รูปที่ 2.แสดงส่วนประกอบของระบบผู้เชี่ยวชาญ

ความสำคัญและหน้าที่ของแต่ละส่วนมีดังนี้

1. ฐานความรู้ (Knowledge Base) เปรียบได้กับฐานข้อมูล (Data base)ในระบบสารสนเทศ (Information system) เป็นส่วนที่ใช้เก็บความรู้ทุกประเภทไม่ว่าจะเป็นความรู้ที่ได้จากตำรา หรือความรู้ที่ได้จากประสบการณ์ ปัญหาหลักของฐานความรู้ก็คือการเลือกวิธีการแสดงความรู้หรือโครงสร้างสำหรับเก็บความรู้ที่เหมาะสม ปัญหานี้เปรียบได้กับการเลือกโครงสร้างข้อมูลหรือโครงสร้างฐานข้อมูลที่เหมาะสมในระบบซอฟต์แวร์ทั่วๆไป

2. เครื่องอนุมาน (Inference Engine) เปรียบได้กับ วิธีการทำงานหรือ อัลกอริธึมที่ใช้ในการคิดค้นหาข้อสรุปและหาเหตุผลหรือวินิจฉัยความจริงใหม่ๆ จากข้อเท็จจริงที่มีอยู่ก่อนแล้ว เป็นส่วนที่ควบคุมการใช้ความรู้ในฐานความรู้เพื่อแก้ไขปัญหาอย่างมีประสิทธิภาพ วิธีการอนุมานมีหลายวิธีแต่แยกเป็นประเภทใหญ่ ๆได้ 2 ประเภท คืออนุมานแบบเดินหน้า (forward chaining inference) และอนุมานแบบย้อนหลัง(backward chaining inference) ทั้งสองวิธีนี้ต่างก็มีจุดดีและจุดเสีย ทั้งนี้ขึ้นอยู่กับลักษณะของปัญหา ในระบบผู้เชี่ยวชาญบางระบบจะใช้ทั้งสองวิธีรวมกัน

3. ส่วนรับความรู้ (Knowledge Acquisition) เป็นส่วนของระบบผู้เชี่ยวชาญที่ใช้ช่วยในการดึงเอาความรู้จากตำรา หรือฐานข้อมูลและจากผู้เชี่ยวชาญ การดึงเอาความรู้จากตำราหรือฐานข้อมูลนั้นทำได้ไม่ยาก ถ้าหากเราสามารถจัดความรู้จากแหล่งดังกล่าวให้เป็นระบบ และเข้ากับได้กับโครงสร้างของฐานข้อมูล เราก็จะสามารถบรรจุความรู้เหล่านั้นเข้าไปในฐานข้อมูลได้ แต่การดึงเอาความรู้จากผู้เชี่ยวชาญนั้นทำได้ยาก จำเป็นต้องใช้เทคนิคต่าง ๆ เข้าช่วย เช่นการสัมภาษณ์หรือการออกแบบสอบถามเป็นต้น หรือไม่ก็ทำให้ระบบผู้เชี่ยวชาญสามารถเรียนรู้ด้วยตนเองในบางส่วนได้

4. ส่วนอธิบาย (Explanation) คือส่วนที่ทำหน้าที่อธิบายรายละเอียดของขั้นตอนการอนุมานต่อผู้ใช้ว่าข้อสรุป หรือคำตอบนั้นได้มาอย่างไรและทำไม

5. ส่วนติดต่อกับผู้ใช้ (User Interface) เป็นส่วนที่เป็นตัวกลางระหว่างผู้ใช้กับระบบ เพื่อทำให้การสื่อสารระหว่างผู้ใช้กับระบบเป็นไปได้อย่างราบรื่น และช่วยให้ผู้ใช้ยอมรับระบบมากขึ้น

เปลือกระบบผู้เชี่ยวชาญและวิศวกรความรู้ (Expert System Shell and Knowledge Engine)

ปัจจุบันระบบผู้เชี่ยวชาญสำเร็จรูปในหัวข้อเรื่องต่างๆที่สามารถนำไปใช้งานตามที่ต้องการได้เลยนั้นยังมีน้อยเนื่องจากความรู้เป็นเรื่องใหญ่และกว้างขวาง ยากที่จะนำความรู้ต่าง ๆ เหล่านั้นมาทำเป็นระบบผู้เชี่ยวชาญสำเร็จรูปในทุกสาขาวิชาได้ ดังนั้นการนำระบบผู้เชี่ยวชาญมาใช้บางครั้งผู้ใช้จะต้องมีการพัฒนาระบบขึ้นมาเอง

การพัฒนาระบบผู้เชี่ยวชาญนั้นจะต้องมีขั้นตอนต่าง ๆ ที่มากมายและซับซ้อน ถ้าหากว่าการพัฒนาระบบผู้เชี่ยวชาญ ต้องเริ่มต้นจากการพัฒนาระบบซอฟท์แวร์ ซึ่งประกอบด้วยเครื่องอนุมาน ฐานความรู้ หน่วยติดต่อกับผู้ใช้และอื่น ๆ ผู้พัฒนาระบบจะต้องเสียเวลามากในการออกแบบ เขียนโปรแกรม และสร้างความรู้ให้ระบบ ในการพัฒนาระบบทางด้านโปรแกรมนั้น ผู้พัฒนาจะต้องรู้เทคโนโลยีทางด้านซอฟท์แวร์ของปัญญาประดิษฐ์ และจะต้องมีความรู้ความชำนาญในเรื่องที่สร้างเป็นระบบผู้เชี่ยวชาญนั้นด้วย

ปัจจุบันเทคโนโลยีทางด้านซอฟท์แวร์ได้ทำให้การพัฒนาระบบผู้เชี่ยวชาญง่ายขึ้น ได้มีการพัฒนาระบบผู้เชี่ยวชาญที่ถูกสร้างให้มีเฉพาะโครงสร้างภายนอกที่เรียกว่า เปลือกระบบผู้เชี่ยวชาญ (Expert System Shell) หรืออีกนัยหนึ่งก็คือตัวโปรแกรมระบบผู้เชี่ยวชาญที่ยังไม่มีข้อมูลความรู้นั่นเอง เปลือกระบบผู้เชี่ยวชาญนี้จะเป็นระบบที่สามารถพัฒนาฐานความรู้ทั่วไปทีหลังตามความต้องการของผู้ใช้ได้ ดังนั้น ในการพัฒนาระบบจริง ๆ ผู้พัฒนามักจะเลือกเอาเปลือกระบบผู้เชี่ยวชาญที่มีอยู่ตามท้องตลาดมาใส่ความรู้ การใส่ความรู้ให้กับระบบหรือจะเรียกอีกนัยหนึ่งว่า การแสดงความรู้ (Knowledge Representation) นั้นเป็นขั้นตอนที่มีความซับซ้อนพอสมควร ผู้ที่ทำหน้าที่แสดงความรู้จะต้องมีความรู้เกี่ยวกับการแสดงความรู้และความรู้เรื่องที่จะใส่ให้กับระบบ ซึ่งผู้ที่ทำหน้าที่เช่นนี้จะเรียกว่า วิศวกรความรู้ (Knowledge Engineer) หน้าที่ของวิศวกรความรู้โดยทั่ว ๆ ไปคือ การศึกษาและนำความรู้จากแหล่งต่าง ๆ เช่น หนังสือ ผู้เชี่ยวชาญ เป็นต้น มาทำการจัดเป็นระบบและสร้างให้อยู่ในรูปแบบที่เหมาะสม เพื่อพร้อมที่จะนำเข้าไปบรรจุในเปลือกระบบผู้เชี่ยวชาญ หน้าที่ที่สำคัญอีกประการหนึ่งของวิศวกรความรู้ ก็คือ การตรวจสอบและบำรุงรักษาระบบ การตรวจสอบนี้จะเป็นการทดสอบการทำงานของระบบผู้เชี่ยวชาญว่าการทำงานเป็นไปตามจุดประสงค์หรือไม่ สำหรับการบำรุงรักษาก็คือ การปรับปรุงฐานความรู้ให้ทันสมัยตามกาลเวลาที่เปลี่ยนไป ซึ่งการทำงานของวิศกรความรู้ตามที่กล่าวมานี้จะต้องประสานกันอย่างใกล้ชิดกับผู้เชี่ยวชาญและแหล่งความรู้ต่าง ๆ

ตัวอย่างระบบผู้เชี่ยวชาญ

เนื้อหาในหัวข้อนี้เป็นการอธิบายถึงระบบผู้เชี่ยวชาญต่างๆที่มีอยู่ในปัจจุบัน (James P. Ignizio 1991 : 51)

-DENDRAL ระบบผู้เชี่ยวชาญสำหรับการวิเคราะห์ทางด้านเคมี

เป็นระบบผู้เชี่ยวชาญในยุคแรกๆ โดยเริ่มพัฒนาในช่วงปี 2505 ที่มหาวิทยาลัยสแตนฟอร์ด ภายใต้การควบคุมของ Joshua Lederberg, Edward Feigenbaum และ Bruce Buchanan, Lederberg เคยได้รับรางวัลโนเบลสาขาเคมี ส่วน Feigenbaum และ Buchanan เป็นผู้ที่ทำงานวิจัยในเรื่องระบบผู้เชี่ยวชาญในยุคแรกๆ ทั้งสามได้ร่วมกันพัฒนาระบบผู้เชี่ยวชาญสำหรับการวิเคราะห์โครงสร้างโมเลกุลของสารประกอบที่ยังไม่เป็นที่รู้จัก ระบบ DENDRAL ได้รับการยอมรับว่าเชื่อถือได้และยังคงมีนักเคมีที่ใช้อยู่ในปัจจุบัน

-HEARSAY I และ II ระบบจดจำเสียงพูด (Speech Recognition)

HEARSAY I เริ่มพัฒนาเมื่อปี 2512 ส่วน HEARSAY II เริ่มพัฒนาในปี 1971 ที่ มหาวิทยาลัยคาเนกี-เมลลอน เพื่อใช้ในงานวิเคราะห์และจดจำเสียงพูด HEARSAY รับข้อมูลให้เป็นรูปคลื่นจากรูปคลื่นที่ได้รับนี้จะนำไปสร้างสมมติฐาน (หลายสมมติฐาน) ว่าเป็นคำพูดอะไร คำตอบที่ดีที่สุดจะถูกส่งออกทางหน่วยแสดงผลลัพธ์ โครงการ HEARSAY ได้พัฒนาจนเสร็จสมบูรณ์ในปี 2518 ระบบนี้สามารถจดจำคำศัพท์ได้ประมาณ 1000 คำ และความถูกต้องในการประมวลผลประมาณ 75 เปอร์เซ็นต์

-MYCIN ระบบผู้เชี่ยวชาญในการวิเคราะห์โรคติดเชื้อทางเลือด

MYCIN เป็นระบบผู้เชี่ยวชาญที่เป็นที่รู้จักมากที่สุดระบบหนึ่งประกอบด้วยกฎประมาณ 500 กฎ ซึ่งการวิเคราะห์ของ MYCIN จะใกล้เคียงกับการวิเคราะห์จากผู้เชี่ยวชาญ

-XCON ระบบผู้เชี่ยวชาญสำหรับการกำหนดส่วนประกอบของเครื่องคอมพิวเตอร์ VAX (Computer Configuration)

XCON พัฒนาขึ้นเพื่อใช้ในการกำหนดส่วนประกอบของเครื่องคอมพิวเตอร์ VAX โดยบริษัทDigital Equipment Cprporation (DEC) โดยร่วมมือกับมหาวิทยาลัยคาเนกีเมลลอน ซึ่งเครื่องคอมพิวเตอร์ VAX สามารถกำหนดส่วนประกอบได้หลายอย่างอย่าง โดยที่ทางบริษัท DEC ได้พยายามที่จะกำหนดองค์ประกอบของเครื่อง VAX ให้เป็นไปตามความต้องการของผู้ใช้มากที่สุด ระบบ XCON ประกอบด้วยกฎประมาณ 8000 กฎ ทำงานบน OPS5 (เป็นระบบที่ทำงานด้วยภาษา LISP และ ใช้การอนุมานแบบเดินหน้า (Forward Chaining)

-GATES ระบบการกำหนดช่องทางสำหรับเครื่องบิน (Gate Assignment)

ระบบนี้ใช้อยู่ที่สนามบินนานาชาติ JFK ที่นิวยอร์ค เพื่อให้คำปรึกษาเกี่ยวกับการควบคุมภาคพื้นดินสำหรับการกำหนดช่องทางสำหรับการลงจอดของเครื่องบิน ฐานความรู้สำหรับระบบนี้

สร้างขึ้นโดยผู้มีประสบการณ์ทางด้านการจัดการภาคพื้นดิน ประกอบด้วยกฎในฐานความรู้ประมาณ 100 กฎ พัฒนาด้วยภาษา PROLOG

การกำหนดช่องทางนั้นเป็นการทำงานที่ซับซ้อนมีตัวแปรที่เกี่ยวข้องหลายอย่าง เช่น สภาพอากาศ เครื่องเสีย การคำนวณผลลัพธ์ในการกำหนดช่องทางใช้เวลาประมาณ 30 วินาที ในขณะที่ถ้าใช้คนในการกำหนดจะต้องใช้เวลาประมาณ 10 ถึง 15 ชั่วโมงในการทำงาน

บทที่ 3
เปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล

หลักการ

เปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูลใช้หลักการของ Client-Server มาประกอบการทำงานของเปลือกระบบผู้เชี่ยวชาญ กล่าวคือ ส่วนที่ทำหน้าที่ประมวลผลความรู้ (ส่วนอนุมาน) จะทำงานในรูปของ Server เพื่อคอยให้บริการแก่โปรแกรมที่ควบคุมการทำงานของอุปกรณ์อินพุทต่างๆ โดยระบบอนุญาตให้ผู้ใช้เขียนโปรแกรมขึ้นมาเชื่อมต่อกับระบบ เพื่อส่งอินพุทหรือรับค่าเอาท์พุทได้ ซึ่งอุปกรณ์อินพุทเหล่านั้นผู้ใช้สามารถกำหนดได้อย่างอิสระว่าจะติดต่อกับอุปกรณ์ใดและเมื่อระบบได้ประมวลผลหาข้อสรุปได้แล้ว ก็จะส่งผลลัพธ์ที่ได้ให้กับโปรแกรมที่ขอใช้บริการ ซึ่งผู้ใช้สามารถกำหนดได้ว่าจะให้ทำอย่างไรกับผลลัพธ์ที่ได้ โปรแกรมที่ผู้ใช้เขียนขึ้นนี้จะผ่านค่าของข้อมูลทาง Message queue โดยในโปรแกรมนี้จะมี subroutine เบื้องต้นที่จำเป็นให้กับผู้ใช้เพื่อให้ง่ายแก่การเขียนโปรแกรมเชื่อมกับระบบ

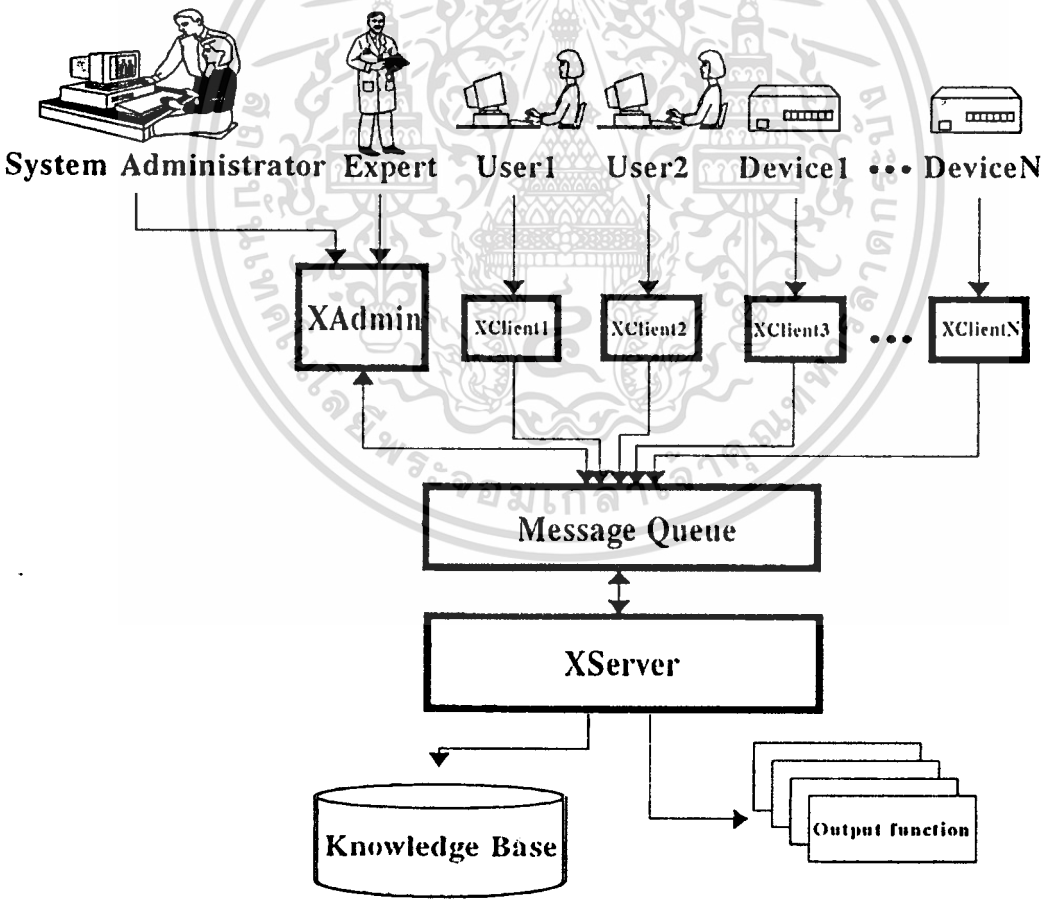
เปรียบเทียบกับหลักการเดิม

สำหรับหลักการเดิมของเปลือกระบบผู้เชี่ยวชาญนั้น ตัวโปรแกรมสำหรับรับส่งข้อมูลจะรวมอยู่ในตัวของเปลือกระบบผู้เชี่ยวชาญ ซึ่งทำให้ผู้ใช้ไม่สามารถจะเปลี่ยนแปลงได้ แต่ในระบบนี้โปรแกรมส่วนที่จัดการการรับส่งข้อมูล, โปรแกรมการจัดการผลลัพธ์และส่วนอนุมานจะทำงานแยกกันโดยอิสระ ผู้ใช้สามารถจะเขียนโปรแกรมของตนเองขึ้นเชื่อมต่อกับระบบได้ โดยมีการส่งค่าผลลัพธ์ของแต่ละโปรแกรมในรูปแบบที่กำหนดไว้

ระบบนี้ จะใช้การติดต่อระหว่างโปรเซสมาช่วยในการทำงานของระบบผู้เชี่ยวชาญ และ

จะมีส่วนควบคุมการติดต่อระหว่างโปรเซสเป็นส่วนที่เชื่อมต่อระหว่างโปรแกรมที่ควบคุมอุปกรณ์รับส่งข้อมูลที่ผู้ใช้เขียนขึ้นกับเปลือกระบบผู้เชี่ยวชาญโดยจะรับผลลัพธ์ที่ได้จากโปรแกรมควบคุมอุปกรณ์รับข้อมูลแล้วนำผลลัพธ์ที่ได้มาส่งให้เปลือกระบบผู้เชี่ยวชาญ เพื่อทำการอนุมาน และเมื่อเปลือกระบบผู้เชี่ยวชาญทำการอนุมานเสร็จก็จะนำผลลัพธ์นั้น ส่งให้กับโปรแกรมที่ผู้ใช้เขียนขึ้นเพื่อให้ โปรแกรมนั้นๆตัดสินว่าจะทำอะไรกับผลลัพธ์ที่ได้ต่อไป นอกจากนี้แล้วผู้ใช้ยังสามารถเรียกใช้โปรแกรมอื่น ในกรณีที่ผลลัพธ์มีค่าตามที่กำหนดไว้ได้ด้วย

ส่วนประกอบของเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล



รูปที่ 3.ส่วนประกอบของเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล

จากรูปที่ 3. แสดงในเห็นส่วนประกอบต่างๆ ของเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล ซึ่งประกอบด้วยส่วนประกอบแยกกันเป็น 3 ส่วน (Program) ใหญ่ๆ คือ

- ส่วนให้บริการของเปลือกระบบผู้เชี่ยวชาญ : XServer (Expert System Shell Server)
- ส่วนควบคุมเปลือกระบบผู้เชี่ยวชาญ : XAdmin (Expert System Shell Administrator)
- ส่วนผู้ใช้บริการเปลือกระบบผู้เชี่ยวชาญ : XClient (Expert System Shell Client)

ซึ่งในแต่ละส่วนมีหน้าที่และส่วนประกอบย่อยๆ ดังนี้

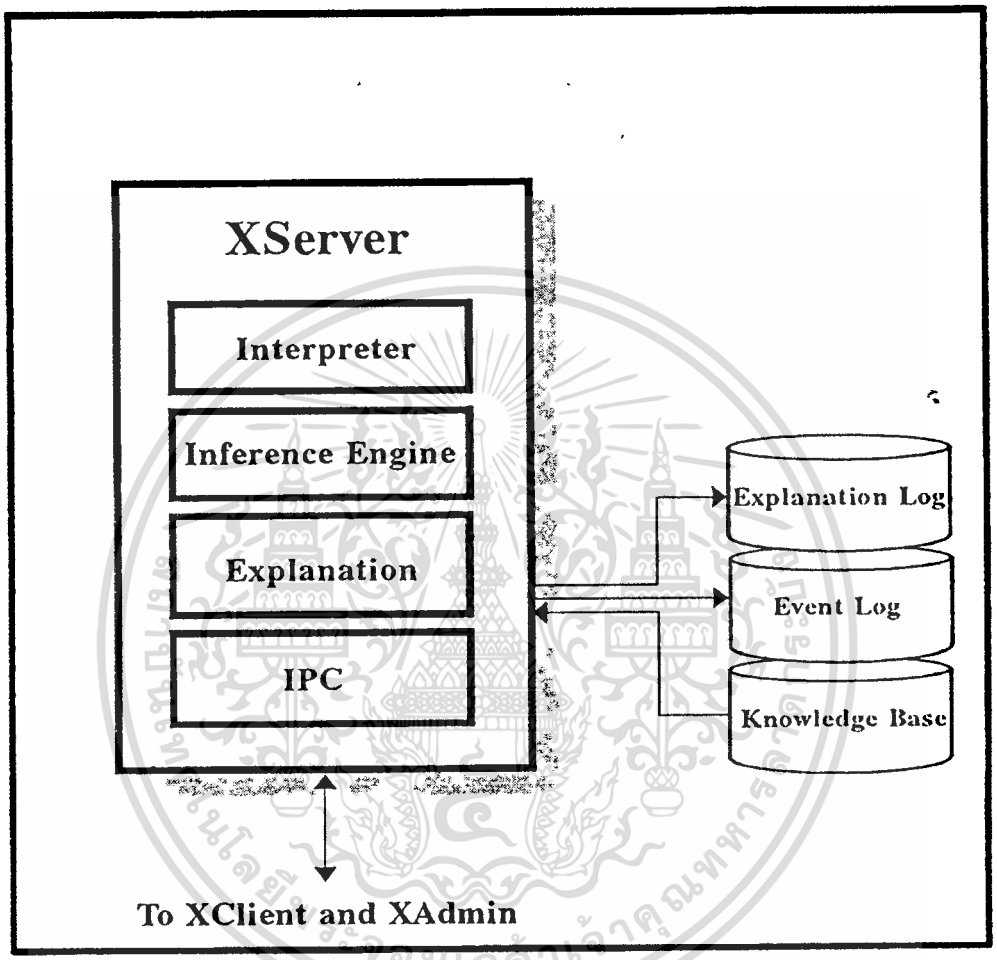
1. XServer ได้แก่โปรแกรมที่ทำงานอยู่เบื้องหลัง (Background Process) เพื่อทำหน้าที่ประมวลผลหรืออนุมาน ของระบบผู้เชี่ยวชาญและให้บริการแก่ XClient ซึ่ง XClient ก็คือโปรแกรมที่คอยควบคุมและรับข้อมูลจากอุปกรณ์อินพุทเอาท์พุทต่างๆ XServer นี้ก็เหมือนกับเปลือกระบบผู้เชี่ยวชาญทั่วๆ ไป ที่ไม่ได้รับข้อมูลผ่านทางแป้นพิมพ์แล้วส่งผลลัพธ์ออกทางจอภาพเหมือนปกติ แต่จะรับข้อมูลและส่งผลลัพธ์ผ่านทาง IPC (InterProcess Communication) แทน เมื่อ XClient ต้องการใช้บริการของ XServer ก็จะทำการส่งข้อความ (Message) ซึ่งมีลักษณะเป็นค่าของตัวแปรตัวอย่างเช่น "COST=100" หรือ "TEMP=high" ไปยัง Message queue ซึ่งเชื่อมต่อกับ XServer ส่วนทางด้าน XServer ก็จะคอยตรวจสอบว่ามีโปรเซสใดต้องการใช้บริการบ้าง เมื่อมี ก็จะอ่านข้อความนั้นเข้ามาแปลความหมายและส่งให้กับส่วนอนุมานเพื่อหาข้อสรุปที่ได้ (จากตัวอย่างคือหาว่า ถ้า COST=100 หรือ TEMP=high จะมีผลอย่างไร) โดยส่วนอนุมานจะคัดเลือกกฎที่เหมาะสมกับ ตัวแปรที่ได้รับมา จากฐานความรู้เข้ามาเพื่อวิเคราะห์หาคำตอบที่เหมาะสม

ส่วน XServer มีส่วนประกอบดังนี้คือ

1.1. ส่วนควบคุมการติดต่อระหว่างโปรเซส (InterProcess Communication Control) เป็นส่วนที่ควบคุมการเชื่อมต่อระหว่าง ส่วน XServer ส่วน XClient และ ส่วน XAdmin โดยที่ส่วน XServer จะเป็นผู้สร้าง Message queue ขึ้นมาเพื่อรอรับข้อมูลจาก XClient และรอรับคำสั่งจาก XAdmin

1.2. ส่วนแปลความหมาย (Interpreter) เป็นส่วนที่ทำหน้าที่ในการตรวจสอบและแปลความหมายของการขอใช้บริการ (Request) จาก XClient หรือ XAdmin แล้วส่งให้กับส่วนอนุมานเพื่อทำการหาข้อสรุปของปัญหา เมื่อได้ข้อสรุปของปัญหาแล้วส่วนอนุมานจะส่งกลับมาให้ส่วนแปลความหมายเพื่อทำการแปลว่าจะต้องทำอย่างไรกับผลลัพธ์ที่ได้ และเนื่องจากเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูลนี้ ผู้ใช้สามารถที่จะสร้างโปรแกรมของตนเองขึ้นมา และสามารถเรียกใช้งานโปรแกรมนั้นๆถ้าเงื่อนไขที่กำหนดไว้เป็นจริงได้ ดังนั้นส่วนแปลความ

หมายนี้จะต้องทำงานร่วมกับส่วนอนุมานกล่าวคือถ้าเงื่อนไขใดที่เป็นจริง (เงื่อนไขในส่วน IF เป็นจริง) ส่วนแปลความหมายจะตรวจสอบส่วนที่เป็นข้อสรุป



รูปที่ 4.แสดงส่วนประกอบของ XServer

(ส่วนที่อยู่หลัง THEN) ว่ามีการเรียกใช้งานโปรแกรมอื่นหรือไม่ ถ้าเป็นการเรียกใช้งานโปรแกรมอื่น ส่วนแปลความหมายก็จะเรียกโปรแกรมนั้นขึ้นมาทำงานต่อไป

1.3. ส่วนอนุมาน (Inference Engine) ทำหน้าที่ในการหาข้อสรุปของปัญหา รูปแบบการอนุมานที่ใช้คือการอนุมานแบบเดินหน้า (Forward Chainning) ซึ่งเป็นการหาข้อสรุปจากตัวแปรหรือสภาวะที่มีการเปลี่ยนแปลง เมื่อมีการเปลี่ยนแปลงตัวแปรหรือสภาวะเกิดขึ้น ส่วนอนุมาน

นี้จะทำการวิเคราะห์ว่าจะได้ข้อสรุปอย่างไร การหาข้อสรุปในแต่ละครั้งอาจทำให้ตัวแปรอื่นเปลี่ยนแปลงค่าได้ ซึ่งเมื่อตัวแปรใหม่มีการเปลี่ยนแปลงค่า เครื่องอนุมานก็จะทำการวิเคราะห์หาข้อสรุปจากผลการเปลี่ยนแปลงค่าของตัวแปรใหม่นั้นจนกว่าจะไม่มีการเปลี่ยนแปลงของตัวแปรอีกจึงจะถือว่าเป็นการสิ้นสุดการอนุมาน และนำข้อสรุปที่ได้ทั้งหมดส่งให้กับส่วนแปลความหมายเพื่อแปลว่าจะต้องทำอย่างไรกับผลลัพธ์ที่ได้ต่อไป

1.4. ส่วนอธิบาย (Explanation) ทำหน้าที่เก็บเส้นทางการค้นหาคำตอบของ ส่วนอนุมานว่าได้ใช้กฎอะไรบ้างในการพิจารณาหาข้อสรุปของปัญหาเพื่ออธิบายให้ผู้ใช้ทราบ

2. ส่วน XAdmin คือส่วนที่ทำหน้าที่ในการควบคุมการทำงานของระบบทั้งหมด ซึ่งได้แก่การ Start, Stop XServer การเพิ่ม การลบ เปลี่ยนแปลง กฎในฐานข้อมูล เมื่อผู้ใช้ต้องการทำงานในส่วนที่เกี่ยวกับการควบคุมการทำงานของระบบ ผู้ใช้สามารถทำได้โดยเรียกโปรแกรม XAdmin ขึ้นมา ส่วน XAdmin นี้ ประกอบด้วยคำสั่งย่อย (Subcommand) ที่สามารถเรียกใช้งานได้หลายคำสั่ง เมื่อผู้ใช้เรียกใช้คำสั่งย่อยใดก็ตาม โปรแกรม XAdmin จะส่ง Control Message ของคำสั่งนั้นๆ ลงใน Message queue เพื่อผ่านไปยัง XServer ที่รอรับอยู่ เนื่องจากตัวโปรแกรม XAdmin สามารถจำลองตัวเองให้ส่งข้อมูลได้แบบเดียวกับ XClient ได้ด้วย ดังนั้นเมื่อ XServer ได้รับ Message ใดๆ เข้ามาก็ตาม XServer จะทำการแปลว่าสิ่งที่ได้รับมาเป็น Control Message หรือข้อมูลที่ต้องการให้ประมวลผล ถ้าเป็น Control Message ก็จะทำงานตามคำสั่งนั้นๆ แต่ถ้าเป็นข้อมูลก็จะส่งให้กับส่วนแปลความหมายและส่วนอนุมานเพื่อทำการสรุปหาคำตอบ

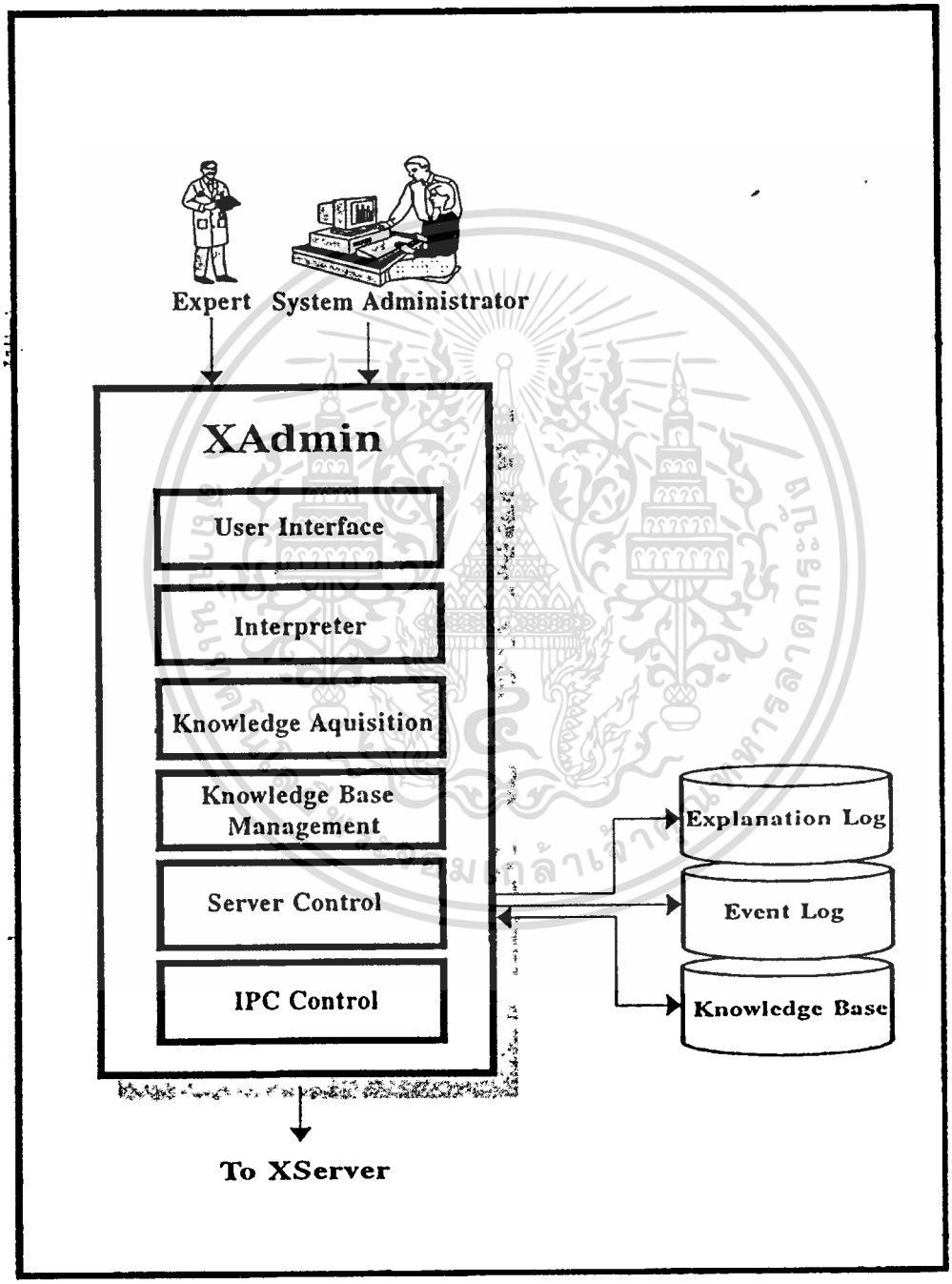
ส่วน XAdmin ประกอบด้วย

2.1. ส่วนรับความรู้ (Knowledge Acquisition) คือส่วนที่สำหรับให้ผู้เชี่ยวชาญใส่ความรู้เข้าในฐานความรู้ การใส่ข้อมูลลงในฐานความรู้นี้ จะทำผ่านทางส่วนติดต่อกับผู้ใช้ จากนั้นส่วนแปลความหมายก็จะทำการตรวจสอบความถูกต้องของกฎที่ใส่เข้าไป ถ้ากฎที่ใส่เข้าไปถูกต้องส่วนรับความรู้ก็จะส่งกฎนั้นๆ ไปให้กับส่วนจัดการฐานความรู้ เพื่อเก็บข้อมูลลงในฐานความรู้ต่อไป

2.2. ส่วนควบคุมการติดต่อระหว่างโปรเซส (InterProcess Communication Control) คือส่วนที่ใช้ในการติดต่อกับ XServer โดยการสร้างทางเชื่อมต่อกับ Message queue ที่ XServer ได้สร้างขึ้น

2.3. ส่วนติดต่อกับผู้ใช้ (User Interface) เป็นส่วนที่รอรับคำสั่งจากผู้ใช้แล้วนำไปส่งให้กับส่วนที่เกี่ยวข้อง และเมื่อส่วนใดต้องการส่งข้อความติดต่อกับผู้ใช้ก็จะส่งกลับมาให้กับส่วนติดต่อกับผู้ใช้ จากนั้นส่วนติดต่อกับผู้ใช้ก็จะส่งข้อความนั้นให้กับผู้ใช้ทางจอภาพ

2.4. ส่วนจัดการฐานความรู้ (Knowledge Base Management) เป็นส่วนจัดการฐานความรู้และให้บริการแก่ส่วนอื่นๆ ที่ต้องการใช้ฐานความรู้ ประกอบด้วย Subroutine หลายๆส่วน ทำหน้าที่ในการ เพิ่ม ลบ เปลี่ยนแปลง การจัดเรียงและการค้นหาข้อมูลในฐานความรู้

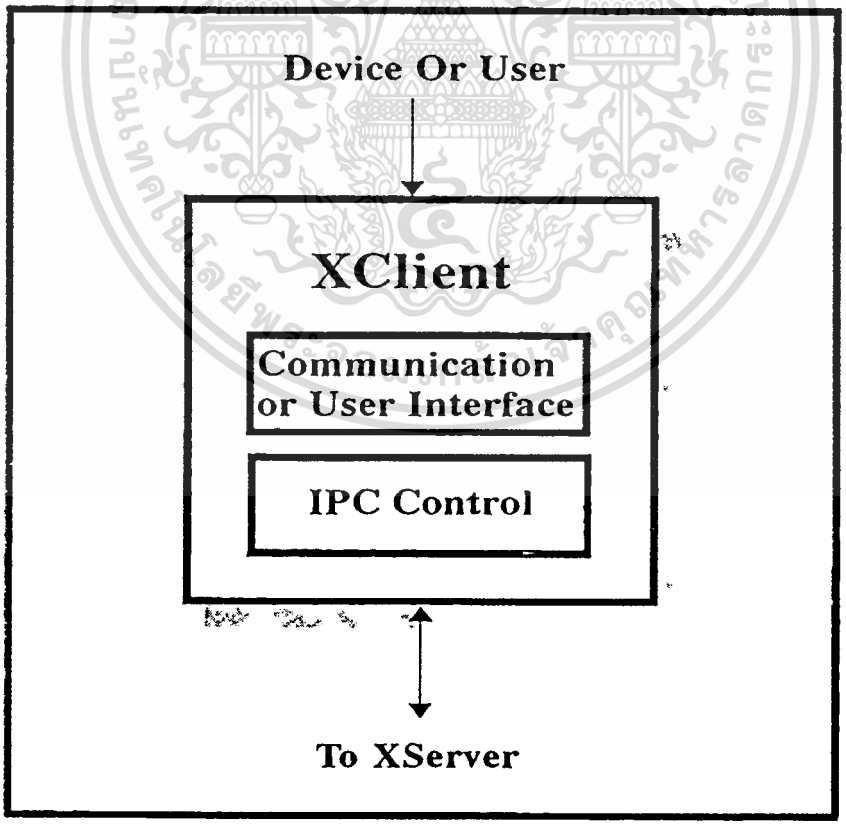


รูปที่ 5.แสดงส่วนประกอบของ XAdmin

2.5. ส่วนแปลความหมาย (Interpreter) ส่วนนี้ทำหน้าที่ในการตรวจสอบและแปลความหมายของ Request ที่รับมาจากส่วนติดต่อกับผู้ใช้ การทำงานจะคล้ายกับส่วนแปลความหมายของ XServer

2.6. ส่วนควบคุมการทำงานของ XServer ส่วนนี้ทำหน้าที่ในการ Start, Stop XServer โดยการ Start XServer จะทำโดยการสร้างโปรเซสของ XServer ให้เป็นการทำงานแบบเบื้องหลัง ซึ่งเมื่อ Start XServer แล้วผู้ใช้สามารถออกจากโปรแกรม XAdmin หรือ exit จากระบบได้โดยไม่มีผลกระทบกับการทำงานของ XServer ส่วนการ Stop นั้นจะทำได้โดยการส่ง Control Message "exit" ให้กับ XServer เมื่อ XServer ได้รับ Message นี้ก็จะหยุดการทำงาน

3. XClient เป็นโปรแกรมที่ทำหน้าที่อ่านข้อมูลจากอุปกรณ์รับข้อมูลซึ่งอาจเป็นได้ทั้ง keyboard, sensor หรืออุปกรณ์อื่นๆแล้วส่งข้อมูลลง Message queue เพื่อเป็น input ให้กับ XServer โปรแกรมของ XClient ที่ใช้งานอยู่ในระบบสามารถใช้งานพร้อมๆกันได้หลายโปรแกรม โดยที่แต่ละโปรแกรมจะส่งข้อมูลไปลงที่ Message queue เดียวกันโดย XServer จะมาอ่านข้อมูลจาก Message queue ตามลำดับก่อนหลัง



รูปที่ 6.แสดงส่วนประกอบของ XClient

ผู้ใช้สามารถที่จะเขียนโปรแกรม XClient ของตนเองขึ้นมาเชื่อมต่อกับระบบได้ โดยใช้ภาษา C ซึ่งจะมี include file และ sub-routine ต่างๆ มาให้เพื่อให้ผู้ใช้สามารถพัฒนาโปรแกรมได้ง่ายขึ้น

สำหรับในบทต่อๆไปจะเป็นการอธิบายส่วนประกอบของระบบซึ่งได้กล่าวมาในหัวข้อต่างๆแล้วในรายละเอียด

บทที่ 4
ส่วนควบคุมเปลือกระบบผู้เชี่ยวชาญ

ส่วนควบคุมเปลือกระบบผู้เชี่ยวชาญ (Expert System Shel Adinstrator Program : XAdmin) เป็นส่วนที่ช่วยให้ผู้ควบคุมระบบทำงานได้อย่างสะดวกขึ้นสามารถใช้ Start, Stop หรือตรวจสอบการทำงานของระบบ หรือทดสอบความถูกต้องของกฎที่ใส่เข้าไปใหม่ได้ นอกจากนี้แล้วผู้เชี่ยวชาญยังสามารถที่จะใส่ความรู้ใหม่ๆเพิ่มให้กับฐานความรู้ของระบบได้โดยผ่านทาง User Interface ที่ระบบมีให้

โปรแกรม XAdmin นี้จะประกอบด้วยส่วนประกอบต่างๆ 5 ส่วนด้วยกันคือ

- ส่วนติดต่อกับผู้ใช้
- ส่วนรับความรู้
- ส่วนควบคุมการติดต่อระหว่างโปรเซส
- ส่วนจัดการฐานความรู้
- ส่วนควบคุมการทำงานของ XServer

สำหรับรายละเอียดในส่วนต่างๆจะได้อธิบายในหัวข้อต่อไปนี้

ส่วนติดต่อกับผู้ใช้ (User Interface)

ส่วนนี้จะเป็นส่วนแรกที่ผู้ใช้ได้พบกับตัวโปรแกรมของโครงงานนี้ โดยที่รูปแบบการทำงานของส่วนนี้จะทำงานในลักษณะ Text Mode และจะมี help สำหรับช่วยเหลือให้ผู้ใช้สามารถใช้คำสั่งได้สะดวกขึ้น รูปแบบการทำงานของส่วน User Interface จะมีดังตัวอย่างข้างล่าง ซึ่งตัวอักษร **ตัวเข้ม** คือคำสั่งที่ผู้ใช้ต้องพิมพ์เข้าไป ส่วน ตัวอักษร *ตัวเอียง* คือข้อมูลที่โปรแกรมแสดงผลออกมา

 # xadmin

Welcome to I/O Independent Expert System Shell

Initialize...

Processing [A.cvl]

Processing [B.cvl]

Processing [C.cvl]

.....

.....

Processing [X.cvl]

Processing [Y.cvl]

Processing [Z.cvl]

Initialize complete

[xAdmin] : **help**

Help Subsystem

delrule	*: Mark a rule deleted*
displayfact	*: Print value of all fact*
help	*: Provide this screen*
loadfact	*: Load default facts from file*
newfact	*: Enter a new fact*
newrule	*: Enter new rule*
udelrule	*: Undelete rule*
packrule	*: Delete all deleted mark rule*
rulelist	*: List all rules*
savefact	*: Save default facts to file*
startsrv	*: Start Expert System Server*
stopsrv	*: Stop Expert System Server*
why	*: Explan how system produce the advice*
xping	*: Ping to XServer*

exit : *Quit from this program*

การทำงานของส่วนติดต่อกับผู้ใช้นี้จะรอรับข้อความจากผู้ใช้ เมื่อได้รับข้อความใดๆเข้ามาส่วนติดต่อกับผู้ใช้ก็จะทำการตรวจสอบว่าข้อความหรือคำสั่งที่พิมพ์เข้ามานั้นถูกหรือไม่ ถ้าผิดก็จะแจ้งให้ ผู้ใช้ทราบแล้ววนกลับมารับคำสั่งจากผู้ใช้ต่อไป ในกรณีที่คำสั่งที่พิมพ์เข้ามาถูกโปรแกรมในส่วนติดต่อกับผู้ใช้นี้ก็จะทำหน้าที่ในการเรียกใช้ module ที่เกี่ยวข้องขึ้นมาทำงาน เมื่อ module นั้นๆทำงานเสร็จแล้วการทำงานก็จะกลับมาอยู่ที่ส่วนติดต่อกับผู้ใช้ จะวนอยู่เช่นนี้จนกระทั่งได้รับคำสั่งให้ออกจากโปรแกรม XAdmin

สำหรับตัวอย่างการใช้งานของส่วนติดต่อกับผู้ใช้จะสามารถดูได้จากบทที่ 7. ในเรื่องตัวอย่างการใช้งาน

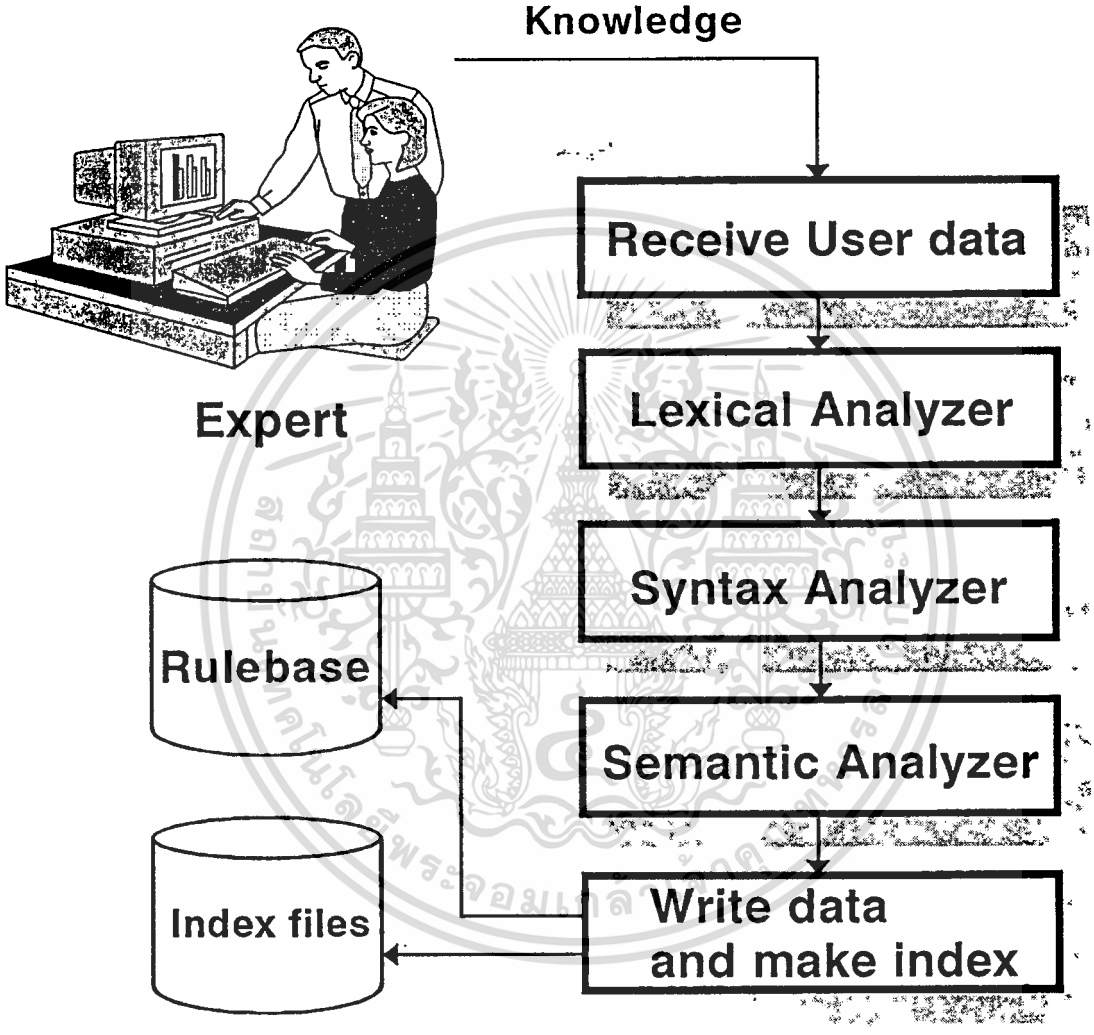
ส่วนรับความรู้ (Knowledge Acquisition)

ส่วนรับความรู้เป็นส่วนที่ทำหน้าที่ในการโต้ตอบกับผู้เชี่ยวชาญหรือวิศวกรความรู้โดยมีจุดมุ่งหมายที่จะให้ผู้เชี่ยวชาญหรือวิศวกรความรู้ผู้นั้นใส่ข้อมูลความรู้ในลักษณะที่เป็นกฎ (rule) หรือข้อเท็จจริง (fact) เมื่อผู้เชี่ยวชาญได้ใส่ข้อมูลแล้วส่วนรับความรู้นี้จะทำหน้าที่ในการนำเอาความรู้ที่ได้ในรูปกฎไปเก็บในฐานความรู้ในรูปแบบที่เหมาะสม

การทำงานของส่วนรับความรู้ในระบบประกอบด้วยส่วนต่างๆดังนี้

- ส่วนรับข้อมูลจากผู้ป้อน
- ส่วน Lexical Analyzer หรือ scanner
- ส่วน syntactic analyzer หรือ parser
- ส่วน Semantic Analyzer
- ส่วนบันทึกข้อมูลและจัดทำดรรชนีสำหรับข้อมูลนั้น ๆ

สำหรับขั้นตอนแสดงการทำงานของส่วนประกอบต่าง ๆ แสดงได้โดยรูปที่ 7.



รูปที่ 7.แสดงขั้นตอนการทำงานของส่วนประกอบต่าง ๆ ในส่วนรับความรู้

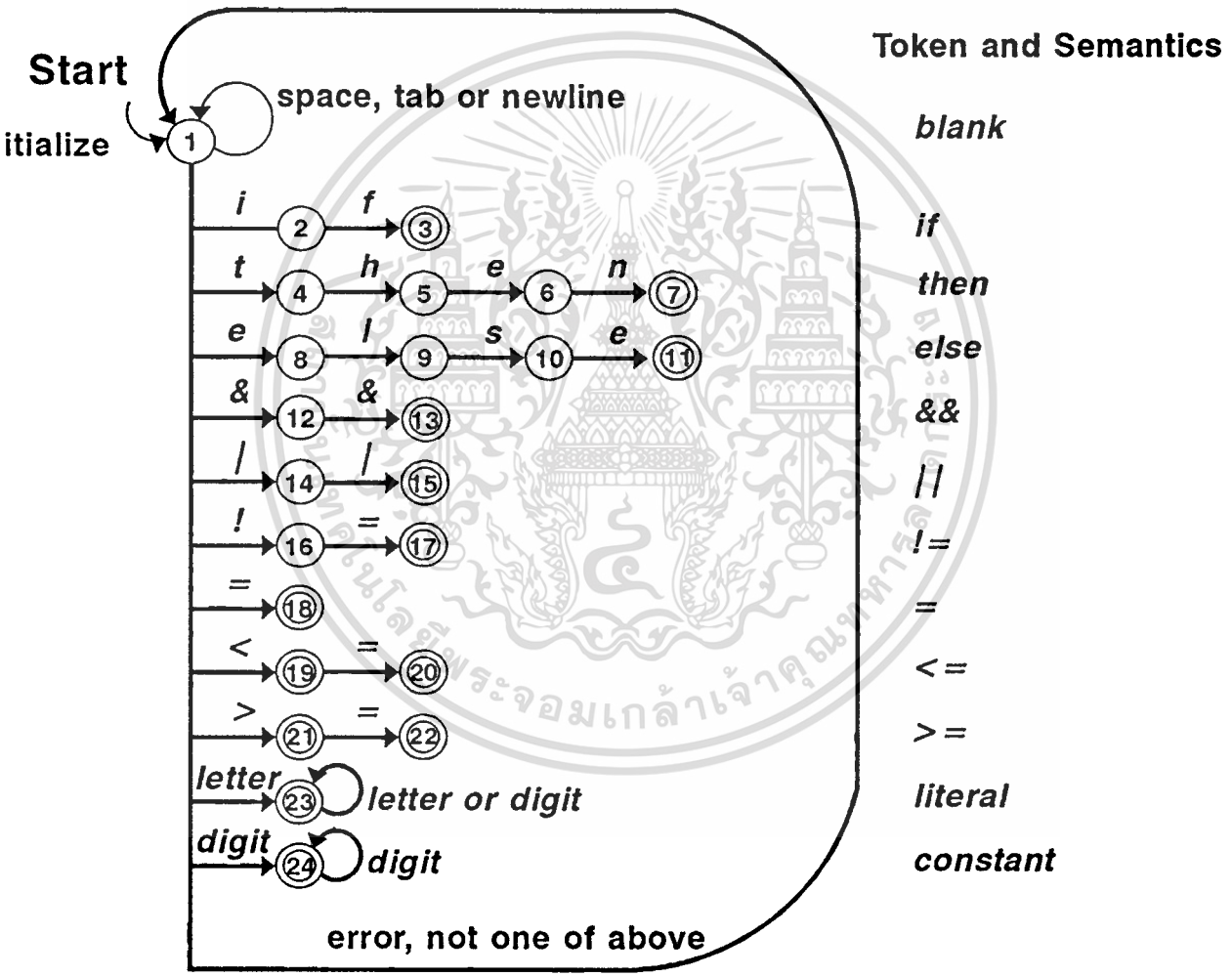
1. ส่วนรับข้อมูลจากผู้ป้อน ส่วนนี้ทำหน้าที่รับข้อมูลจากแป้นพิมพ์และจะทำการส่งต่อให้กับส่วน Lexical Analyzer นำไปใช้งานต่อไป ซึ่งข้อมูลที่ได้จากส่วนรับข้อมูลก็คือกฎต่าง ๆ ที่ผู้

เชี่ยวชาญใส่ลงไป ส่วนรับข้อมูลจะถูกเรียกใช้งานอีกครั้งเมื่อได้ประมวลผลกฎที่ใส่เข้าไปหลังสุดเสร็จ และจะวนรับข้อมูลจนกว่าผู้ใช้พิมพ์คำว่า "exit" จึงจะจบกระบวนการการใส่ข้อมูล

2. ส่วน Lexical Analyzer หรือ scanner

คือส่วนที่ทำหน้าที่เชื่อมต่อระหว่างส่วนรับข้อมูลและส่วน Syntax Analyzer ซึ่งส่วน Lexical Analyzer นี้จะทำงานโดยพิจารณากฎที่ผู้เชี่ยวชาญใส่เข้าไป ทีละตัวอักษร จากนั้นก็จะแยกข้อกฎนี้เป็นส่วนย่อย ๆ เรียกว่า token ซึ่งแสดงถึง ชื่อตัวแปร, operator, คำสั่งต่าง ๆ รวมถึงนิพจน์ทางคณิตศาสตร์ นอกจากนี้แล้ว Lexical Analyzer ยังต้องทำหน้าที่ตรวจสอบอักขระและคำต่าง ๆ ของกฎที่ใส่เข้าไปว่าถูกต้องหรือไม่ เช่น ผู้เขียนโปรแกรมเขียน token ผิดหรือมีอักขระที่ไม่อยู่ในกลุ่มของอักขระที่กำหนด และยังจัดการกับช่องว่างขาว (white apace) เช่น การเว้นวรรค tab หรือการขึ้นบรรทัดใหม่ด้วย

Syntax Analyzer จะรับข้อมูลที่ได้จาก Lexical Analyzer เพื่อนำไปประมวลผลต่อ โดยในทางปฏิบัติ Syntax Analyzer สามารถนำกฎที่ผู้เชี่ยวชาญใส่เข้าไปซึ่งเป็นภาษาที่มนุษย์เข้าใจง่ายไปประมวลผลได้โดยไม่ต้องใช้ Lexical Analyzer เป็นตัวช่วยก็ได้ แต่ token ต่าง ๆ จะมีความซับซ้อนซึ่งจะทำให้การสร้าง Syntax Analyzer ทำได้ยากและซับซ้อน รวมทั้ง Syntax Analyzer ที่ไม่มีส่วน Lexical Analyzer จะใช้เวลาในการประมวลผลนาน และใช้ทรัพยากรของระบบมาก ด้วยสาเหตุนี้จึงทำให้ต้องมีการนำเอา Lexical Analyzer มาเพื่อปรับปรุงโครงสร้างข้อมูลที่ผู้ใช้ใส่เข้าไปเพื่อให้มีรูปแบบที่ง่ายขึ้น



รูปที่ 8. แสดง Finite state diagram ของ Lexical Analyzer ของระบบ

ตัวอย่างข้างล่างเป็นตัวอย่างการกำหนดการแปลงข้อความต่างๆในกฎให้เป็น token โดยใช้ภาษาซี

```
#define IF          (unsigned char)125      /* if     */
#define THEN        (unsigned char)126      /* then   */
#define ELSE        (unsigned char)127      /* else   */
#define CALL        (unsigned char)128      /* call   */
#define AND         (unsigned char)140      /* &&     */
#define OR          (unsigned char)141      /* ||     */
#define NOT         (unsigned char)142      /* !      */
#define EQ          (unsigned char)143      /* =      */
#define NEQ         (unsigned char)144      /* !=     */
#define GT          (unsigned char)145      /* >      */
#define GTE         (unsigned char)146      /* >=     */
#define LT          (unsigned char)147      /* <      */
#define LTE         (unsigned char)148      /* <=     */
```

การทำงานของ Lexical Analyzer จะเริ่มต้นจากการรับกฎจากส่วนรับข้อมูลจากผู้ป้อน ซึ่งตัวอย่างของกฎที่ผู้ใช้ใส่เข้าไปจะเป็นดังข้อความข้างล่าง

```
if (ANT=1)&&(BAT=1) then (CAT=1)&&(call(func1 arg1 arg2))
```

จากนั้นจะทำการอ่านและตีความทีละอักขระเริ่มจากด้านซ้ายมือก่อน โดยการตีความจะใช้วิธีการตามรูปที่ 8. ซึ่งเมื่อเสร็จกระบวนการของ Lexical Analyzer แล้ว จะได้ผลลัพธ์ดังตัวอย่างข้างล่าง (ตำแหน่ง 0100 ถึง 0126)

```
45D2:0100  7D 28 41 8F 31 29 8C 28-42 8F 31 29 7E 28 43 8F  }(A.1).(B.1)~(C.
45D2:0110  31 29 8C 28 80 28 66 75-6E 63 31 20 61 72 67 31  1).(.(func1 arg1
45D2:0120  20 61 72 67 32 29 29 1A-B6 0D 16 C2 16 C0 16 F8   arg2))........
```

```
45D2:0130  8E C2 00 00 AC 8A D0 4E-AD 8B C8 46 8A C2 24 FE  .......N...F..$.
45D2:0140  3C B0 75 05 00 55 AC F3-AA EB 06 3C B2 75 6D 6D  <.u..U.....<.umm
45D2:0150  80 A2 13 A8 01 74 B1 BE-32 01 8D 8B 1E 90 40 8E  .....t..2.....@.
45D2:0160  FC 33 D2 29 E3 13 8B C2-03 C3 15 00 69 0B F8 83  .3.).........i...
45D2:0170  FF FF 74 11 26 01 1D 00-A0 E2 F3 81 FA 00 F0 74  ..t.&..........t
```

สำหรับตัวโปรแกรมการทำงานของ Lexical Analyzer จะมีตัวอย่างดังโปรแกรมข้างล่าง

```
10   #include <stdio.h>
20   #include "lex.h"
30   lexical_analyzer(char *istr, char *lexstr)
40   { int  i = 0;       /* counter for input string          */
50     int  j = 0;       /* counter for output string         */
60     while(istr[i] != '\0')
70     { switch(istr[i])
80       { case ('i') : if(istr[i+1] == 'f')
90                   {  lexstr[j++] = IF;
100                  i += 2;          /* skip 'if' */
110                  }
120                  else lexstr[j++] = istr[i++];
130                  break;
140                        case ('t') : if(istr[i+1] == 'h' && istr[i+2] == 'e' && istr[i+3]
== 'n')
150                  {  lexstr[j++] = THEN;
160                    i += 4;
170                  }
180                else lexstr[j++] = istr[i++];
190                  break;
200          case ('e') : if(istr[i+1] == 'l' && istr[i+2] == 's' && istr[i+3] == 'e')
```

```
210                     {  lexstr[j++] = ELSE;
220                       i += 4;
230                     }
240                     else lexstr[j++] = istr[i++];
250                     break;
260           ....
270   }
```

จากตัวอย่างโปรแกรมข้างต้น เป็นการผ่านค่ากฎให้ส่วน Lexical Anlayzer ทำการวิเคราะห์ โดยที่ข้อมูลความรู้ที่ต้องการจะตรวจสอบความถูกต้องจะอยู่ในตัวแปร "istr" และผลลัพธ์ที่ได้จากการทำงานของส่วนนี้จะถูกเก็บไว้ในตัวแปร "lexstr" โปรแกรมส่วน Lexical Analyzer จะทำการ Scan ตัวอักษรทีละตัวโดยเริ่มจากซ้ายไปขวาเช่นถ้าพบตัวอักษร "i" ก็จะทำการตรวจสอบตัวถัดไปว่าเป็น "f" หรือไม่ถ้าใช่ในนำค่า "IF" ซึ่งคือ (unsigned char)125 ตามที่เราได้ define เอาไว้ใน "lex.h" ไปใส่ใน "lexstr" และทำการตรวจสอบความถูกต้องของคำต่างๆที่มีในกฎที่ใส่เข้าไป โดยในส่วน Lexical Analyzer นี้ยังไม่สนใจเรื่องลำดับของ Keyword ต่างๆ เช่นในกรณีที่มี "then" มาก่อน "if" ส่วนนี้จะยังไม่สามารถตรวจสอบได้ว่ามีความผิดพลาดซึ่งการตรวจสอบนี้จะถูกทำโดยส่วน Syntax Analyzer ที่จะได้กล่าวถึงในหัวข้อถัดไป

ตัวโปรแกรมเต็มของ Lexical Analyzer จะดูได้จากภาคผนวก ข. ในโปรแกรม "lex.c"

3. ส่วน Syntax Analyzer หรือ Parser เป็นส่วนที่มีความซับซ้อนกว่า Lexical Analyzer โดยส่วนนี้จะเป็นส่วนที่ทำการตรวจสอบความถูกต้องของโครงสร้างข้อมูล โดยจะเปรียบเทียบได้กับการตรวจสอบโครงสร้างของประโยคในภาษาต่าง ๆ ว่า ตำแหน่งของประธาน กริยา กรรม และส่วนขยายอื่น ๆ อยู่ในตำแหน่งที่ถูกต้องหรือไม่ ในตัวแปลภาษา (compiler) ทั่ว ๆ ไป ส่วนนี้จะทำการสร้าง Syntax Tree เพื่อส่งต่อให้ Semantic Analyzer ประมวลผลต่อไป แต่ในโครงงานนี้ข้อความต้นแบบมีความง่ายกว่าไวยากรณ์ของภาษาต่าง ๆ มาก จึงไม่เหมาะที่จะสร้างเป็น Syntax Tree โดยส่วน Syntax Analyzer จะทำหน้าที่เพียงแค่ตรวจสอบความถูกต้องของโครงสร้างของชุดของ token ที่ได้จาก Lexical Analyzer จากนั้นก็จะส่งชุดของ token ไปให้ Semantic Analyzer ถ้าพบข้อผิดพลาด (error) ก็จะแสดงข้อความออกทางหน้าจอให้ผู้ใช้ทราบ กรณีที่ token ไม่เรียงกันเป็นประโยคตามที่กำหนด (เช่น then เกิดขึ้นก่อน if) Syntax Analyzer จะต้องสามารถตรวจจับความผิดพลาดและดำเนินการต่อให้ได้ เพื่อที่จะได้ตรวจจับความผิดพลาดครั้งต่อไปที่อาจจะเกิดขึ้นได้อีก ซึ่งคอมไพเลอร์ที่เป็นที่ยอมรับจะไม่หยุดการตรวจสอบเมื่อพบความผิดพลาดครั้งแรก

ตัวอย่างข้างล่างคือตัวอย่างโปรแกรมของ Syntax Analyzer

```
10    while(istr[i] != '\0')
20      { switch (istr[i])
30        {  case (IF) :  if(i != 0)       /* if must place at 1st char */
40                              {  err("[SYN] : if much be located at the first character
in rule\n");
50                                 if_flag++;
60                                 i++;
70                                 ret -= 1;
80                              }
90                              else
100                             {  if_flag++;
110                                i++;
120                             }
130                             break;
140             case (THEN): if(if_flag == 0)  /* no "if" before "then"     */
150                             {  err("[SYN] : No if before then\n");
160                                then_flag++;
170                                i++;
180                     ret -= 1;
190                             }
200                             else
210                             {  i++;
220                                then_flag++;
230                             }
240                             break;
250
260                     ...
270   }
```

โดยในโปรแกรมจะรับค่าอินพุทมาจากตัวแปรชื่อ "istr" แล้วนำมาตรวจสอบทีละขั้นโดย Scan จากซ้ายไปขวา และจะมีตัวแปรต่างๆคอยเก็บสถานะ เช่น "if_flag" ถ้ามากกว่า 0 แสดงว่า ค่าตัวอักษรที่กำลังตรวจสอบอยู่นั้นอยู่ในตำแหน่งหลัง "if" แล้ว และถ้าค่า "then_flag" มีค่าเป็น 0 แสดงว่าตัวอักษรที่กำลังตรวจสอบอยู่นั้นอยู่หน้า "then" เป็นต้น แต่ถ้าตรวจสอบพบ "then" ((unsigned char)126) ในขณะที่ตัวแปร "if_flag" มีค่าเป็น 0 ก็แสดงว่า ในมีการพบว่า มี "then" อยู่หน้า "if" ซึ่งส่วน Syntax Analyzer จะทำการแจ้ง Error Message ออกมา

ส่วน Syntax Analyzer นี้จะทำการตรวจสอบการวางตำแหน่งของ Keyword ต่างๆว่าวางอยู่ในตำแหน่งที่ถูกต้องหรือไม่เท่านั้น แต่จะไม่รู้ความหมายของ Keyword ต่างๆเลย และยังไม่รู้ว่าจะต้องทำอะไรกับตัวแปรต่างๆ ซึ่งส่วนนี้จะถูกวิเคราะห์โดยส่วน Semantic Analyzer ซึ่งจะกล่าวถึงในหัวข้อถัดไป

สำหรับแผนภาพการทำงานของส่วน Syntax Analyzer จะอยู่ในภาคผนวก ก. และตัวโปรแกรมเต็มจะอยู่ในภาคผนวก ข. ในโปรแกรมชื่อ "syntax.c"

4. ส่วน Semantic Analyzer ส่วน Semantic Analyzer ทำหน้าที่พิจารณาความหมายของกฎที่ผู้เชี่ยวชาญใส่เข้าไป ถึงแม้ว่าในทางทฤษฎีแล้วจะมีการแยก Semantic Analyzer กับ Syntax Analyzer ออกจากกัน แต่ในการทำงานจริง ทั้งสองส่วนจะมีส่วนที่ทำงานร่วมกัน เช่น expression (A+B) Semantic Analyzer จะพิจารณาเครื่องหมายทางคณิตศาสตร์ว่าเครื่องหมาย "+" หมายความว่าจะต้องนำ A และ B มาบวกกันเป็นต้น ส่วน syntax analyzer จะเรียก semantic routine ซึ่ง routine นี้จะทำการตรวจสอบว่าตัวแปร A และ B กำหนดค่ามาแล้วหรือยังและเป็นชนิดเดียวกันหรือไม่

Semantic Analyzer ในส่วนรับความรู้ (knowledge acquisition) นี้จะนำข้อมูลที่ผู้ใช้ใส่ให้กับระบบมาพิจารณาความหมายเพื่อตรวจสอบความหมายของข้อมูลว่ามีความหมายถูกต้องหรือไม่ เพื่อหลีกเลี่ยงการเกิดข้อผิดพลาดที่อาจจะเกิดขึ้นได้ในตอนประมวลผล แต่จะไม่มีการคำนวณค่าของตัวแปรต่าง ๆ

ตัวอย่างข้างล่างต่อไปนี้เป็นตัวอย่างโปรแกรมในส่วนของ Semantic Analyzer โดยใน Module นี้จะรับข้อมูลในรูป (Value1, Operator, Value2) เช่น calculate (VALUE1, "EQ", VALUE2) จะเป็นการคำนวณว่า ค่าของ VALUE1 เท่ากับค่าของ VALUE2 หรือไม่ถ้าเท่าก็จะ return ค่าเป็น 1 ถ้าไม่เท่าจะ return ค่าเป็น 0

```
10    int calculate(char *VALUE1, char oper, char *VALUE2)
```

```
20   {
30     switch(oper)
40     { case AND:  return((int)VALUE1 && (int)VALUE2);
50       case OR :  return((int)VALUE1 || (int)VALUE2);
60       case EQ :  if(strcmp(VALUE1, VALUE2) == 0)
70                     return(1);
80                  else
90                     return(0);
100      case NEQ:  if(strcmp(VALUE1, VALUE2) != 0)
110                    return(1);
120                 else
130                    return(0);
140      case GT :  if(strcmp(VALUE1, VALUE2) > 0)
150                    return(1);
160                 else
170                    return(0);
180      case GTE:  if(strcmp(VALUE1, VALUE2) >= 0)
190                    return(1);
200                 else
210                    return(0);
220      case LT :  if(strcmp(VALUE1, VALUE2) < 0)
230                    return(1);
240                 else
250                    return(0);
260                 ......
270  }
```

ถ้าเรามีตัวแปรที่ต้องการอนุมานคือ "COST=500" และเรามีกฎคือ "if COST<1000 then BUY=yes" ส่วน Semantic Analyzer จะทำหน้าที่ตรวจสอบว่า "COST<1000" เป็นจริงหรือไม่

โดยในส่วน Interpreter จะทำการอ่านกฎนี้ขึ้นมาและแบ่งกฎนี้เป็นส่วน If-Clause และส่วน Then Clause ซึ่งเราจะได้

ส่วน If-Clause คือ COST<1000

ส่วน Then-Cluase คือ BUY=yes

จากนั้นจะทำการแทนค่าตัวแปร "COST" ในส่วน If-Clause ด้วย "500" และแทนเครื่องหมาย "<" ด้วย "LT" ซึ่งค่าของ "LT" นี้ได้มีการกำหนดไว้แล้วใน File "lex.h" ว่าเป็น (Unsigned char)147 แล้วส่งผ่านค่านี้ในกับ Function "calculate" ข้างต้น จะได้ "calculate (500, LT, 1000)" ซึ่งเป็นการหาคำตอบว่า "500< 1000" หรือไม่ ถ้าน้อยกว่าจะ Return ค่าเป็น 1 โดยเมื่อพิจารณาจาก Function "calculate" จะพบ

```
case LT :  if(strcmp(VALUE1, VALUE2) < 0)
               return(1);
           else
               return(0);
```

จะได้ว่า "500 < 1000" เป็นจริง ดังนั้นจะ Return "1" เมื่อส่วน Interpreter ได้รับคำตอบว่าส่วน If-Clause มีการ Return ค่าเป็น "1" ก็จะนำส่วน Then-Clause มาปฏิบัติซึ่งก็คือการแทนค่าตัวแปร "BUY" ด้วยค่า "yes" ซึ่งก็จะเป็นการจบการทำงานของ ส่วน Semantic Analyzer

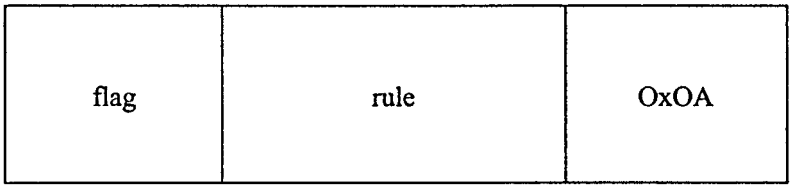
สำหรับแผนภาพการทำงานของ Semantic Analyzer จะอยู่ใน ภาคผนวก ก.

5. ส่วนบันทึกข้อมูลและจัดทำดรรชนีสำหรับกฎใหม่

เมื่อข้อมูล (rule) ที่ผู้ใช้ใส่เข้ามาในระบบได้รับการตรวจสอบโดย Lexical Analyzer, Syntax Analyzer และ Semantic Analyzer แล้ว ข้อมูลที่ได้จะถูกบันทึกลงในแฟ้มข้อมูลชื่อ rbase.xsrv ซึ่งจะเก็บในรูปของ token (ผลลัพธ์ที่ได้จาก Lexical Analyzer) เพื่อความสะดวกในการนำมาประมวลผล และจะประหยัดเนื้อที่ในการเก็บข้อมูล (เช่น "if" จะถูกเก็บเป็น (unsigned char)125 ซึ่งจะใช้เนื้อที่เก็บ 1 ตัวอักษร) ซึ่งก่อนที่จะบันทึก rule นั้นลงแฟ้มข้อมูลนั้นได้ทำการใส่ flag ของ rule นั้น ๆ ลงไปที่ตัวอักษรแรกของข้อมูลดังรูปที่ 9. เพื่อเป็นตัวบอกสถานะของ rule นั้น ซึ่งสถานะของ rule จะมีดังนี้

D = Delete

U = Usable



รูปที่ 9. แสดงการเก็บ กฎ และ flag แสดงสถานะของกฎ

เนื่องจากข้อมูลที่ได้ในรูปของกฎต่างๆนั้นจะถูกนำมาใช้ต่อในช่วงของเครื่องอนุมาน (Inference Engine)ซึ่งในวิทยานิพนธ์นี้ใช้การอนุมานแบบเดินหน้า (forward chaining) การทำงานของเครื่องอนุมานแบบเดินหน้านี้จะต้องใช้การค้นหากฎโดยใช้ตัวแปรในส่วนของ IF เป็น key ในการค้นหา ดังนั้นเราจึงต้องจัดเรียงข้อมูลที่จะบันทึกในรูปแบบที่สามารถค้นหาได้โดยใช้ตัวแปรเป็น key ได้อย่างสะดวกรวดเร็ว ในวิทยานิพนธ์ฉบับนี้ใช้วิธีสร้าง index file ซึ่งเรียกว่า Clause Variable List file ขึ้นมาเพื่อทำหน้าที่บอกตำแหน่งของกฎที่มีตัวแปรนั้นปรากฏอยู่ในส่วน if ตัวอย่างเช่น

เมื่อเราใส่กฎ "if((ANT=1)&&(BAT=2))then(CAT=3) ลงไปใน rule base ในขั้นแรกโปรแกรมจะทำการตรวจสอบความถูกต้องของกฎที่ใส่เข้ามา จากนั้นก็จะพิจารณาว่าในส่วนของ if มีตัวแปรอะไรบ้างซึ่งจากตัวอย่างข้างต้นจะพบว่ามีตัวแปร "ANT" และ "BAT" ดังนั้นก่อนที่โปรแกรมจะเขียนกฎนี้ลงในฐานความรู้โปรแกรมจะทำการสร้าง index โดยจะอ่านค่าตำแหน่งที่จะเขียนกฎใหม่ลงในฐานความรู้ แล้วทำการแก้ไขข้อมูลใน Clause Variable List จากนั้นจึงทำการจัดเรียงข้อมูลใน Clause Variable List ใหม่ โดยเรียงตามลำดับชื่อตัวแปรตามลำดับจากน้อยไปมาก (จาก A ไป Z) แล้วจึงเขียนกฎใหม่คือ if ((ANT=1) && (BAT=2)) then CAT=3) ลงในฐานความรู้

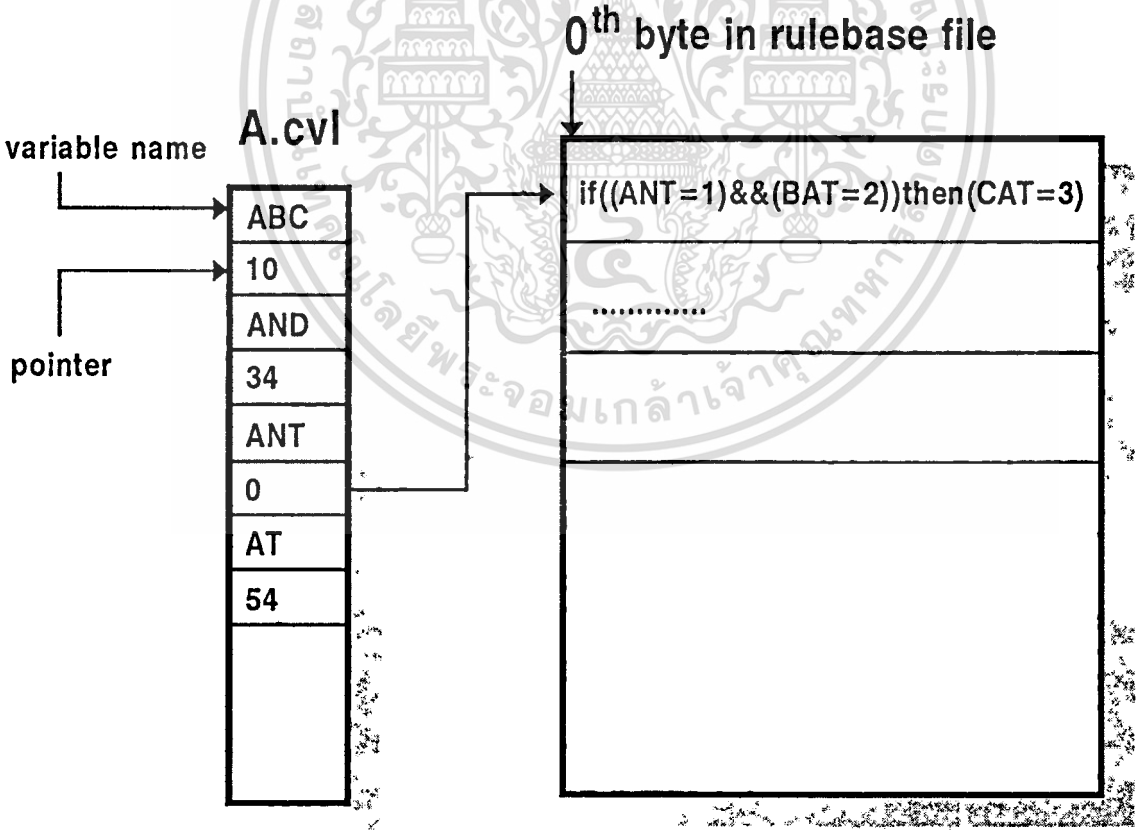
Clause Variable List file จะมีทั้งหมด 26 file โดยมีส่วนขยายเป็น .cvl ชื่อ file จะตามตัวอักษรภาษาอังกฤษจาก "A.cvl" ถึง "Z.cvl" ตัวแปรที่ขึ้นต้นด้วยตัวอักษรใดก็จะถูกเก็บอยู่ใน Clause Variable Llist file ตามตัวอักษรนั้น เช่น ตัวแปร "ANT" และ "BAT" จะเก็บอยู่ใน file "A.cvl" และ "B.cvl" ตามลำดับซึ่งข้อมูลใน Clause Variable List file จะถูกจัดเรียงใหม่ตามลำดับตัวอักษรเสมอ

ในการณีที่มีกฎใหม่เข้ามาในระบบ ถ้ากฎที่จะเก็บใหม่นี้เป็นกฎแรกในฐานความรู้ค่าตำแหน่งที่จะเขียนกฎคือ "0" ดังนั้นใน A.cvl จะมีข้อมูลดังนี้

```
# cat A.cvl
ANT
0
```

และ ใน B.cvl จะมีข้อมูล

```
# cat B.cvl
BAT
0
```



รูปที่ 10. แสดงความสัมพันธ์ของ Clause Variable List file และ ฐานความรู้

ส่วนจัดการฐานความรู้ (Knowledge Base Management)

ฐานความรู้ เป็นส่วนหนึ่งที่สำคัญที่สุดส่วนหนึ่งของระบบผู้เชี่ยวชาญ เพราะเป็นที่สำหรับเก็บความรู้ทั้งหมดของระบบ โดยทั่ว ๆ ไปภายในฐานความรู้จะมีข้อมูลความรู้ 2 ชนิดเก็บอยู่คือ ความจริง (fact) และกฎ (rule), ความจริง (fact) คือความรู้ที่ปรากฏอยู่ก่อนแล้ว ส่วนกฎ (rule) เป็นความรู้ที่ได้จากความเชี่ยวชาญของผู้เชี่ยวชาญที่สร้างระบบผู้เชี่ยวชาญนั้น ๆ (heuristic) ซึ่งความรู้ชนิดนี้ทำให้ระบบผู้เชี่ยวชาญสามารถคาดเดาอย่างมีหลักการในการแก้ปัญหาจากข้อมูลที่ผิดพลาด หรือข้อมูลที่ไม่สมบูรณ์ การจัดการเก็บความรู้ทั้งสองประเภทนี้เป็นงานหลักของส่วนจัดการฐานความรู้

ฐานความรู้นี้เป็นที่สำหรับเก็บและจัดการความรู้ในรูปแบบที่เปลือกระบบผู้เชี่ยวชาญนั้นสามารถนำไปใช้ได้ง่ายและมีประสิทธิภาพ ซึ่งความรู้นี้เปรียบได้กับข้อมูลที่อยู่ในฐานข้อมูล (Database) ของโปรแกรมต่างๆ แต่ความรู้จะแตกต่างจากข้อมูลในจุดหลักสองประการคือ ข้อมูลจะแสดงข่าวสาร (information) อย่างชัดเจนเช่น "ธนาคารคิดดอกเบี้ยร้อยละ 12 ต่อปี" แต่ประโยค "ถ้าดอกเบี้ยสูงหุ้นจะตก" เป็นความรู้เพราะว่าคำว่า "ดอกเบี้ยสูง" นั้นไม่ได้บอกอย่างชัดเจนว่าคำว่า "สูง" นั้นมีค่าเท่าไร ในความคิดของคนสามารถรู้ได้โดยง่ายเช่น ปกติดอกเบี้ยเป็นร้อยละ 10 แต่เปลี่ยนเป็นร้อยละ 12 คนเราจะรู้ได้ว่าดอกเบี้ยสูงขึ้น ที่เป็นเช่นนี้เพราะมนุษย์เรารู้ข่าวสารที่ซ่อนรูปอยู่ แต่การที่จะให้คอมพิวเตอร์รู้ว่าเมื่อใดดอกเบี้ยสูงนั้นทำได้ไม่ง่าย ปัญหาจึงมีอยู่ว่าจะบันทึกส่วนของความรู้ที่ซ่อนรูปอยู่ได้อย่างไร

ประการที่สองความรู้แตกต่างจากข้อมูลตรงที่ว่าความรู้ส่วนใหญ่แสดงออกในรูปของภาษาธรรมชาติ (Natural language) เช่น ภาษาไทย ภาษาอังกฤษ ดังนั้นก่อนที่เราจะสามารถสร้างคอมพิวเตอร์ที่สามารถประมวลผลความรู้ได้ เราจำเป็นที่จะต้องทำให้ความรู้นี้อยู่ในรูปแบบที่คอมพิวเตอร์เข้าใจได้ง่ายก่อน ในหัวข้อนี้จะกล่าวถึงการเปลี่ยนความรู้ให้เป็นแบบที่คอมพิวเตอร์สามารถนำไปใช้งานได้ การแสดงความรู้ในรูปของกฎ และการจัดเก็บความรู้ของโครงงานนี้

1. ความรู้และข้อมูล ความรู้ที่มนุษย์เรามีอยู่นั้นมีหลายรูปแบบถึงแม้เราจะไม่รู้ว่าความรู้เหล่านี้ถูกเก็บอยู่ในสมองในรูปโครงสร้างแบบใด แต่การที่จะทำให้คอมพิวเตอร์สามารถใช้ความรู้ได้ เราจำเป็นจะต้องบันทึกความรู้ในรูปแบบโครงสร้างใดโครงสร้างหนึ่งเข้าไปในคอมพิวเตอร์ ปัญหานี้เรียกว่า การแสดงความรู้ การแสดงความรู้นี้เป็นหัวใจสำคัญของการสร้างระบบความรู้

ความรู้ถึงแม้จะมีอยู่หลายแบบ แต่พอจะแยกออกเป็นสี่ประเภทใหญ่ๆ ได้ดังนี้

- ความรู้ที่บอกความจริง, ลักษณะหรือคุณสมบัติ เช่น ที่ดินผืนนี้กว้าง 100 ตารางวา

- ความรู้ที่บอกความสัมพันธ์ เช่น นกเขาเป็นสัตว์ปีกชนิดหนึ่ง หรือแขนเป็นอวัยวะส่วนหนึ่งของมนุษย์

- ความรู้ที่บอกขั้นตอนหรือวิธีการ เช่น ถ้าหากอุณหภูมิห้องสูงกว่า 40 องศาเซลเซียส ให้ปิดเครื่อง

- ความรู้ที่เกี่ยวกับความรู้ (meta knowledge) เช่น ความรู้เกี่ยวกับคุณลักษณะของความรู้อื่น หรือเกี่ยวกับวิธีการใช้ความรู้อื่น

ความรู้ประเภท 1 และ 2 นั้นได้จากความจริงที่เป็นรูปธรรมความรู้ประเภท 3 เป็นความรู้ประเภทที่อาจจะเรียกได้ว่า "อัลกอริธึม" ส่วนความรู้ประเภท 4 นั้นอาจจะเข้าใจยาก ดังนั้นจะขอยกตัวอย่างประกอบสองตัวอย่างเพื่อช่วยให้เข้าใจง่ายขึ้น ตัวอย่างแรกเป็นตัวอย่างของความรู้เกี่ยวกับคุณลักษณะของความรู้ สมมติว่ามีฐานความรู้หนึ่งมีข้อมูลความรู้เกี่ยวกับเงินเดือนของพนักงานในบริษัท A (ความรู้ประเภท 1) และความสัมพันธ์ระหว่างพนักงานในบริษัทนั้น (ความรู้ประเภทที่ 2) ว่าใครเป็นหัวหน้างาน ความรู้ที่ว่าเงินเดือนของพนักงานที่เป็นหัวหน้าจะสูงกว่าพนักงานที่เป็นลูกน้องถือได้ว่าเป็น meta knowledge สำหรับฐานความรู้ เพราะบอกความสัมพันธ์ระหว่างความรู้เกี่ยวกับเงินเดือน

วิธีการแสดงความรู้ที่ดีนั้นจะต้องมีคุณสมบัติดังต่อไปนี้

- มีสมรรถภาพในการแสดงความรู้ชนิดต่างๆได้ และถ้าเป็นไปได้โครงสร้างที่ใช้ในการแสดงความรู้จะต้องเป็นโครงสร้างที่ง่าย แต่มีสมรรถภาพในการแสดงความรู้สูง

- มี modularity กล่าวคือ ความสามารถในการแยกออกเป็นข้อย่อย (module) ทั้งนี้เพื่อให้สามารถเพิ่มหรือแก้ไขฐานความรู้ได้

- ง่ายต่อการจัดการ กล่าวคือ เป็นคุณสมบัติที่ช่วยในการตรวจสอบฐานความรู้ อย่างเช่นช่วยในการตรวจดูความขัดแย้งในความรู้ การซ้ำกัน หรือความผิดพลาดในความรู้

- ง่ายต่อความเข้าใจของมนุษย์ การแสดงความรู้ที่ดีนอกจากเข้ากับคอมพิวเตอร์ได้แล้วยังจะต้องให้เข้ากับมนุษย์ได้ดีด้วยคุณสมบัติอันนี้ช่วยทำให้การสร้างส่วนอธิบายในระบบผู้เชี่ยวชาญง่ายขึ้นนอกจากนั้นยังช่วยในการตรวจความผิดพลาดในการพิมพ์ความรู้เข้าไปในฐานความรู้ด้วย

- เข้ากันได้กับการอนุมาน ทั้งนี้เนื่องจากการอนุมานต้องใช้ความรู้ในฐานความรู้เป็นข้อมูลดังนั้นเพื่อที่จะให้มีการอนุมานที่มีประสิทธิภาพดี การแสดงความรู้จะต้องเข้ากันได้ดีกับการอนุมาน

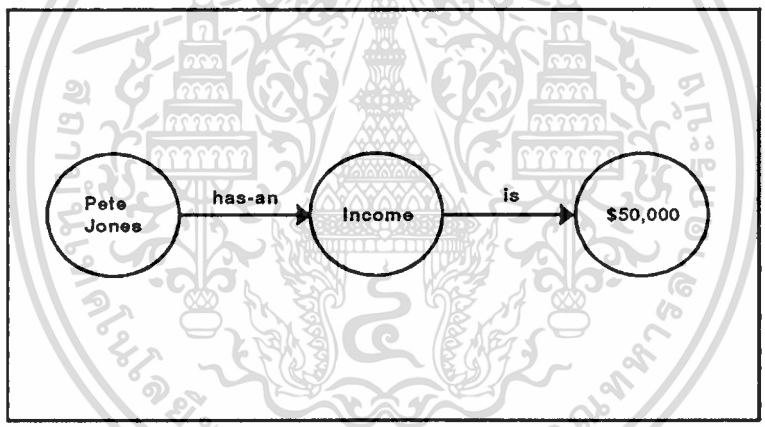
2. การแสดงความรู้

สำหรับหัวข้อต่อไปนี้เป็นการแสดงความรู้ที่นิยมใช้ในระบบผู้เชี่ยวชาญ (Jame P. Ignizio 1991 : 69)

2.1. OAV (Object Attribute Value) Triplets เป็นการแสดงความรู้โดยการแสดงค่าของคุณสมบัติ (attribute) ของวัตถุเป้าหมาย (object) นั้นๆว่ามีค่า (Value) เป็นอย่างไร เช่น ถ้าเราสนใจเครื่องบิน เราจะได้ว่าคุณสมบัติของเครื่องบินคือ

- จำนวนเครื่องยนต์
- ชนิดของเครื่องยนต์ (เครื่องบินใบพัดหรือเครื่องบินไอพ่น)
- รูปแบบของปีก (ปีกแบบธรรมดาหรือแบบลู่ไปทางท้าย)

การแสดงความรู้ในรูปของ OAV ยังมีได้อีกรูปแบบหนึ่งคือแสดงในรูปของ nodes และ branches ดังตัวอย่างในรูปที่ 11.



รูปที่ 11.แสดงการแสดงความรู้ในรูป OAV โดยใช้ nodes และ branches

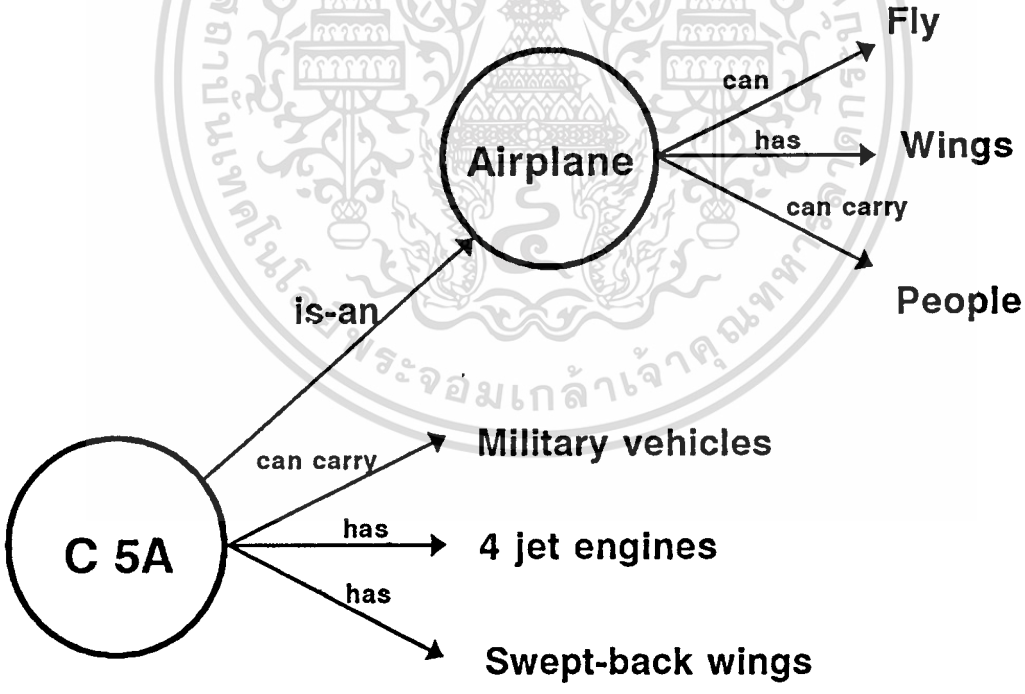
จากรูปที่ 11. จะได้ว่าวัตถุเป้าหมาย คือ Pete Jones คุณสมบัติคือ income และ มีค่าเป็น $50,000

2.2. ข่ายความหมาย (Semantic Networks) คิดค้นขึ้นโดย M.R. Quillian ในปี 1968 (.วิลาส วูวงศ์, บุญเจริญ ศิริเนาวกุล 1992 : 30) เพื่อเป็นแบบจำลองความคิดของมนุษย์ เนื่องจากเป็น

แบบจำลองของขบวนการจดจำของมนุษย์ ดังนั้นจึงมีการนำเอาข่ายความหมายมาใช้ในการแสดงความรู้ แล้วเก็บเข้าไปในคอมพิวเตอร์

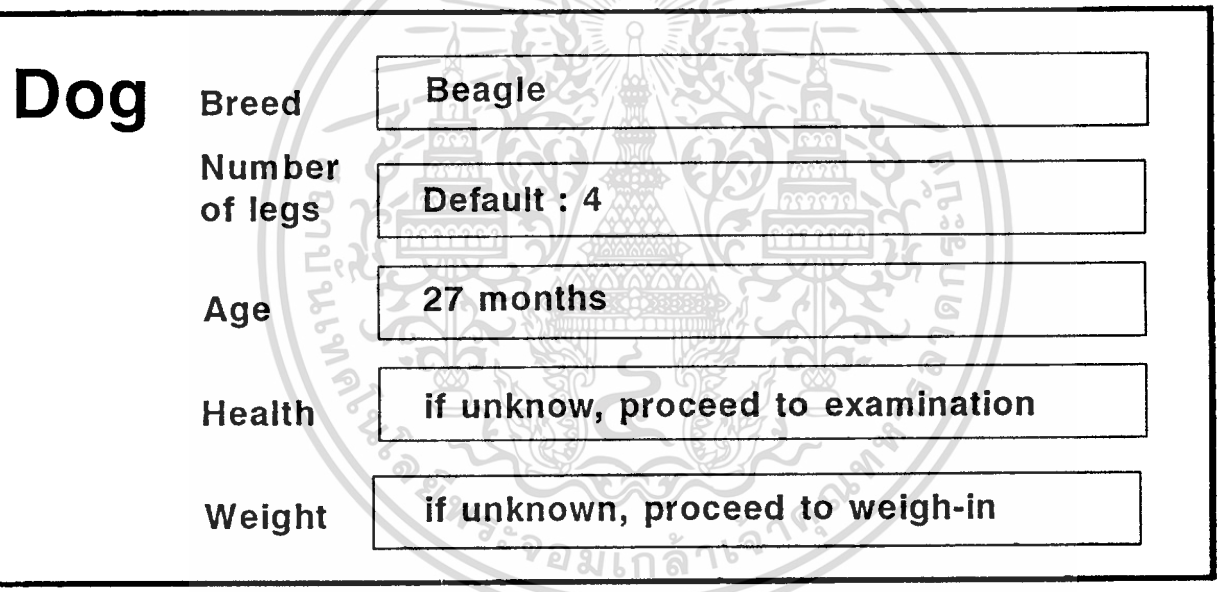
ข่ายความหมาย จะมีลักษณะการนำเอา OAV Triplets มาเชื่อมโยงกันในลักษณะที่ซับซ้อนขึ้นซึ่ง ข่ายความหมายนี้อาจจะใช้แสดงวัตถุหลายวัตถุ และแต่ละวัตถุอาจจะมีคุณสมบัติได้หลายๆข้อ รูปที่ 12. แสดงตัวอย่างการแสดงความรู้โดยใช้ ข่ายความหมาย ของเครื่องบิน C5A

จากรูปที่ 12.แสดงคุณสมบัติของเครื่องบิน C5A ว่าสามารถบรรทุกยานพาหนะได้ มีเครื่องยนต์ 4 เครื่อง และมีปีกแบบลู่ไปด้านท้าย นอกจากนี้แล้วการแสดงความรู้แบบ ข่ายความหมาย ยังสามารถแสดงการถ่ายทอด (inherits) คุณสมบัติได้ กล่าวคือเมื่อได้แสดงคุณสมบัติของ C5A ว่าเป็นเครื่องบินแล้ว C5A จะมีคุณสมบัติเหมือนเครื่องบินคือ บินได้ มีปีก และสามารถบรรทุกผู้โดยสารได้ด้วย



รูปที่ 12.แสดงการแสดงความรู้ด้วยข่ายความหมาย

2.3. กรอบ (Frames) เป็นรูปแบบการแสดงความรู้ที่เสนอขึ้นโดย M.Minsky ในปี 1974 เพื่อเป็นโครงสร้างในการสร้างแบบจำลองของความจำและขบวนการเรียนรู้ของมนุษย์ กรอบเป็นการแสดงความรู้แบบโครงสร้างชนิดหนึ่ง ในกรอบจะมีการบันทึกข้อมูลเกี่ยวกับสถาพเหตุการณ์ วัตถุ หรือความคิด และการบันทึกความสัมพันธ์ต่างระดับระหว่างสิ่งต่างๆเหล่านั้น รูปที่ 13. แสดงตัวอย่างของกรอบ



รูปที่ 13.แสดงการแสดงความรู้ด้วยกรอบ

2.4 ข้อความตรรก (Logic Statements) คือข้อความทางตรรกศาสตร์ที่อาจเป็นได้ทั้งจริงหรือเท็จ ขึ้นอยู่กับค่าสถานะของตัวแปรที่ใส่เข้าไปตัวอย่างของ ข้อความตรรกคือ

Mammal(dog) หมายถึง หมาเป็นสัตว์เลี้ยงลูกด้วยนมซึ่งเป็นจริง แต่ถ้าแทน dog ด้วย fish คือ

Mammal(fish) ข้อความนี้จะเป็นเท็จ

2.5 Rule Based Systems เป็นวิธีที่นิยมมากที่สุดในการแสดงความรู้ในระบบผู้เชี่ยวชาญ (James ginozio 1991 : 74) ความรู้จะถูกแทนด้วยประโยค IF - THEN ตัวอย่างเช่น

IF (engine_type = jet) and (wing_position = low) THEN plan = B747

เป็นการแสดงคุณสมบัติของเครื่องบิน B747 โดยใช้การแสดงความรู้แบบกฎ การแสดงความรู้ในรูปแบบกฎนี้เป็นแบบที่ง่ายต่อความเข้าใจของมนุษย์ และง่ายต่อการสร้างเป็นฐานความรู้

3. การแสดงความรู้ในเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล

ในโครงงานนี้เลือกเอาการแสดงความรู้ในรูปของกฎมาใช้ในส่วนฐานข้อมูลเนื่องจากว่าการแสดงความรู้ในรูปแบบของกฎมีข้อดีดังนี้

- ง่ายต่อความเข้าใจของมนุษย์ ผู้พัฒนา ผู้ใช้ระบบ ใช้เวลาในการเรียนรู้หรือเข้าใจน้อย
- การสร้างฐานความรู้ทำได้ง่าย มีความซับซ้อนน้อยไม่ต้องการฮาร์ดแวร์เพิ่ม
- ฐานความรู้ที่แสดงความรู้ในรูปแบบของกฎ จะง่ายในการปรับปรุงแก้ไขฐานความรู้

ในหัวข้อต่อจากนี้จะอธิบายถึงการเก็บและการแปลงความรู้ในรูปของกฎเพื่อเก็บในฐานความรู้ของโครงงานนี้

3.1 รูปแบบของกฎที่ใช้ในโครงงานมีรูปแบบดังนี้

if (ส่วนเงื่อนไข) then (ส่วนข้อสรุป) else (ส่วนข้อสรุป)

ตัวอย่างเช่น

if (DOLLAR=fall) then (INTEREST = rise) else (INTEREST = fall)

3.1.1. ส่วนเงื่อนไข คือประโยคที่ต้องการตรวจสอบค่าของตัวแปรที่ต้องการว่าเป็นเช่นไรโดยถ้าเป็นจริงจะนำข้อความในส่วนข้อสรุปหลัง then มาปฏิบัติแต่ถ้าเป็นเท็จจะนำข้อ

ความในส่วนข้อสรุปหลัง else มาปฏิบัติ คำว่า if, then และ else เป็นคำสงวน (reserved word) จะต้องเขียนด้วยอักษรภาษาอังกฤษตัวเล็กเสมอ

รูปแบบของส่วนเงื่อนไขมีดังนี้

(VARIABLE operator VALUE)

โดยที่

VARIABLE คือชื่อของตัวแปร โดยที่ชื่อของตัวแปรจะต้องเป็นอักษรภาษาอังกฤษตัวใหญ่เสมอ

operator คือตัวดำเนินการทางคณิตศาสตร์ ดังนี้คือ
เท่ากับ (=)
ไม่เท่ากับ (!=)
มากกว่า (>)
มากกว่าหรือเท่ากับ (>=)
น้อยกว่า (<)
น้อยกว่าหรือเท่ากับ (<=)

VALUE คือค่าของตัวแปลนั้น จะมีค่าเป็นตัวอักษรหรือตัวเลขก็ได้

ตัวอย่างที่ถูกต้องของส่วนเงื่อนไขมีดังนี้

(TEMP = 100)
(AGE >= 12)
(LENGTH < 20)
(STR = hello)

เราสามารถใช้เครื่องหมายทางตรรกศาสตร์ AND (&&), OR (||) เชื่อมส่วนเงื่อนไขสองส่วนได้ตัวอย่างเช่น

(TEMP = 100) && (LENGTH < 20)

(SIZE > 120) || (COST < 50)

3.1.2. ส่วนข้อสรุป เป็นส่วนที่บอกให้ทราบว่าจะต้องทำอย่างไรถ้าผลลัพธ์ของส่วนเงื่อนไขเป็นจริงในกรณีที่ส่วนข้อสรุปอยู่หลัง if และบอกว่าจะต้องทำอย่างไรถ้าผลลัพธ์ของส่วนเงื่อนไขเป็นเท็จในกรณีที่ส่วนข้อสรุปอยู่หลัง then รูปแบบของส่วนเงื่อนไขมีดังนี้

(VARIABLE = VALUE)

โดยที่

VARIABLE คือชื่อของตัวแปร โดยที่ชื่อของตัวแปรจะต้องเป็นอักษรภาษาอังกฤษตัวใหญ่เสมอ

VALUE คือค่าของตัวแปรนั้น จะมีค่าเป็นตัวอักษรหรือตัวเลขก็ได้

ตัวอย่างที่ถูกต้องของส่วนเงื่อนไขมีดังนี้

(COST = 100)

(GRADE = A)

(RETURN = goodby)

นอกจากนี้แล้วในเปลือกระบบผู้เชี่ยวชาญโครงงานนี้ยังมีฟังก์ชัน "call" ที่สามารถใส่ในส่วนข้อสรุปได้หน้าที่ของฟังก์ชัน "call" คือ ใช้สำหรับเรียกใช้งานโปรแกรมอื่นหรือคำสั่งของยูนิกซ์ที่ต้องการ ซึ่งผู้ใช้สามารถเขียนโปรแกรมขึ้นมาเองเพื่อให้ระบบเรียกใช้ได้ รูปแบบการใช้งานฟังก์ชัน "call" คือ

call (โปรแกรมที่ต้องการเรียกใช้)

ตัวอย่างเช่น

call (/home/user1/prog1)

call (rm *.txt)

call (ftp 150.1.1.200)

เราสามารถใช้เครื่องหมายทางตรรกศาสตร์ AND (&&), OR (||) เชื่อมส่วนข้อสรุปหลายๆส่วนหรือใช้เชื่อมระหว่างข้อสรุปและฟังก์ชัน "call" ได้ตัวอย่างเช่น

(GRADE = A) && (SCORE = 20)

(COUNT = 120) || (COST = 50)

(VAR1 = aaa) && call (mkdir temp)

ตัวอย่างของกฎที่ถูกต้องมีดังนี้

if (TEMP = 100) then (RESULT = 50)

if (AGE >= 12) then (PRICE = 20)

if (SIZE > 100)&&(COST < 50) then (RESULT = buy)

if (DISKSPACE < 10)&&(BUFSIZE > 20) then call (rm *.out) && (SRVFLAG = stop)

3.2. รูปแบบการเก็บกฎของระบบลงในแฟ้มข้อมูล เมื่อผู้ใช้ต้องการใส่ความรู้ (กฎ) ลงในฐานความรู้ โปรแกรมรับความรู้จะทำการตรวจสอบว่ากฎที่ใส่เข้ามาในระบบนั้นมีไวยากรณ์และความหมายที่ถูกต้องหรือไม่ เพราะการใส่กฎที่ผิดไวยากรณ์และความหมายเข้าไปนั้นอาจทำให้โปรแกรมทำงานผิดปกติได้ โดยรายละเอียดของส่วนที่ทำการตรวจสอบความถูกต้องประกอบด้วยส่วน Lexical Analyzer ทำหน้าที่ในแปลงกฎให้อยู่ในรูปแบบที่คอมพิวเตอร์สามารถเข้าใจได้ง่าย, Syntax Analyzer ทำหน้าที่ตรวจสอบไวยากรณ์ของกฎที่ใส่เข้าไป, Semantic Analyzer ทำหน้าที่ตรวจสอบความหมายของกฎว่าถูกต้องหรือไม่ รายละเอียดของส่วนตรวจสอบความถูกต้องของกฎนี้ได้กล่าวถึงในรายละเอียดในหัวข้อ 4.1 เรื่องส่วนรับความรู้

เมื่อได้ตรวจสอบความถูกต้องแล้วกฎที่ผ่านการตรวจสอบนี้จะถูกบันทึกลงในแฟ้มข้อมูลชื่อ rbase.xsrv โดยแฟ้มข้อมูล rbase.xsrv มีข้อมูลดังนี้

cat rbase.xsrv

U{(DOLLAR=fall)}(INTEREST=rise)

U{(DOLLAR=rise)}(INTEREST=fall)

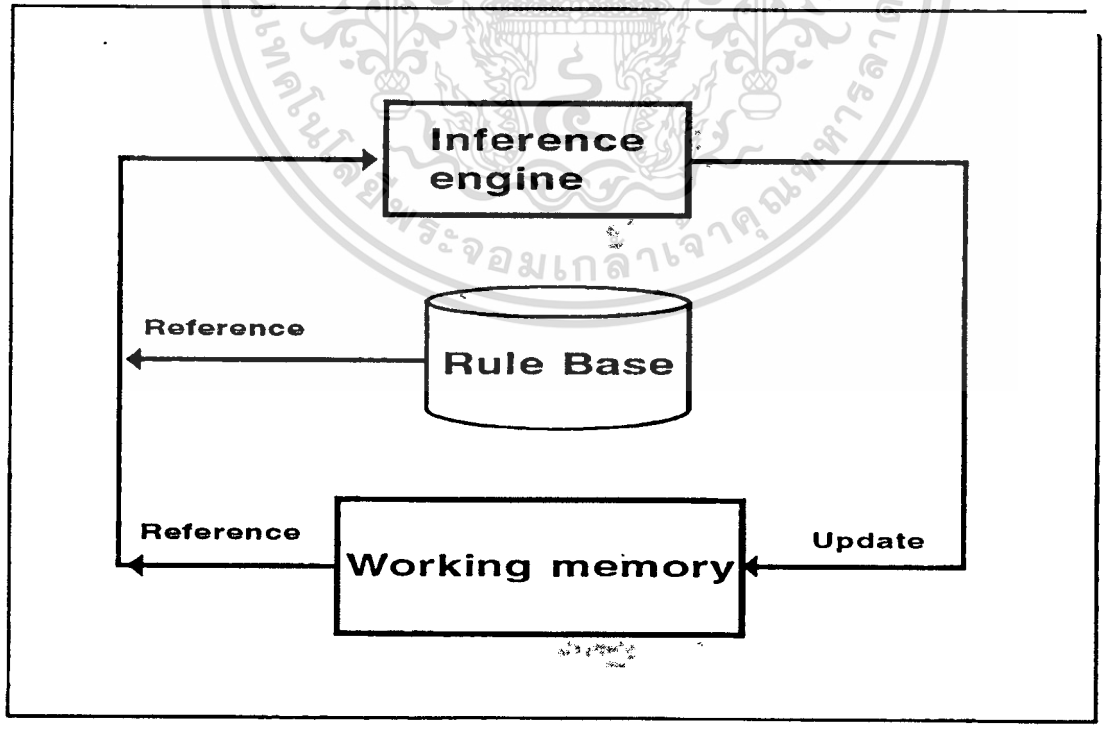
U{(A=1)}(B=1)

U{(B=1)}(C=1)

U{(C=1)}(D=1)

เนื่องจากกฎที่เก็บไว้นี้จะต้องถูกนำไปใช้ในขั้นตอนของ การอนุมาน (inference) โดยที่ในโครงงานนี้ใช้การอนุมานแบบเดินหน้า ซึ่งเป็นการหาข้อสรุปจากเหตุที่เกิดขึ้น เมื่อมีการเปลี่ยนแปลงของตัวแปร เราจะต้องนำตัวแปรนั้นๆไปหาในส่วน เงื่อนไขของกฎทุกกฎว่าจะทำให้กฎใดถูกใช้งานบ้าง ซึ่งในการทำการค้นหานั้นถ้าเราค้นหาทีละกฎว่ากฎใดมีตัวแปรนั้นอยู่ในส่วนเงื่อนไข ก็จะเสียเวลามาก ดังนั้นในโครงงานนี้จึงสร้างดรรชนีขึ้นมาเพื่อที่จะเป็นเครื่องชี้ว่าตัวแปรนั้นอยู่ในกฎไหนบ้าง ทำให้เราไม่ต้องเสียเวลาในการค้นหากฎทั้งหมด โดยที่ ดรรชนีที่สร้างขึ้นมานั้นจะมีลักษณะเป็นแฟ้มข้อมูลซึ่งจะเรียกว่า Clause Variable List หรือแฟ้มข้อมูลที่มีส่วนขยายเป็น .cvl ดังที่ได้กล่าวมาแล้ว

3.3. โครงสร้างข้อมูลสำหรับเก็บสถานะของตัวแปรในฐานความรู้ (working memory)



รูปที่ 14.แสดงโครงสร้างของการแสดงความรู้ในรูปของกฎ

การแสดงความรู้ในรูปของกฎประกอบด้วยโครงสร้างข้อมูลหลักสามส่วนด้วยกันคือ ฐานความรู้ ส่วนอนุมาน และ ส่วน working memory

ฐานความรู้ เป็นที่เก็บความรู้ที่อยู่ในรูปของกฎ ข้อมูลใน working memory นั้นเป็นส่วน input ของส่วน if ของกฎและจะถูกอ้างอิงและเปลี่ยนแปลงโดยกฎในฐานความรู้ โดยที่ส่วนอนุมานนั้นจะทำหน้าที่ในการเลือกกฎใดกฎหนึ่งจากเซ็ตของกฎที่มีเงื่อนไขครบขึ้นมาปฏิบัติ

ในโครงงานนี้ ส่วน working memory ทำหน้าที่ในการเก็บค่าของตัวแปรที่มีอยู่ในกฎต่างๆในฐานความรู้ โดยที่การเปลี่ยนแปลงค่าใน working memory นี้เริ่มต้นจากการเปลี่ยนแปลงที่ได้รับมาจากส่วนรับข้อมูล เมื่อมีการเปลี่ยนแปลงใน working memory ส่วนอนุมานจะนำค่าตัวแปรที่เปลี่ยนแปลงนี้ไปอนุมานจากฐานความรู้ว่าจะมีผลอย่างไรซึ่งผลที่ได้นี้อาจทำให้ตัวแปรอื่นมีการเปลี่ยนแปลงค่าได้ เมื่อผลของการอนุมานตัวแปรแรกทำให้ตัวแปรอื่นเปลี่ยนแปลงไปส่วนอนุมานก็จะนำตัวแปรที่เปลี่ยนแปลงไปนี้มาทำการอนุมานต่อจนกว่าจะสิ้นสุดคือไม่มีการเปลี่ยนแปลงค่าของตัวแปรอีก

โครงสร้างข้อมูลใน working memory นี้ประกอบด้วยตัวแปร array สามชุดคือ

3.3.1.vartab เป็นตัวแปรอักขระทำหน้าที่เก็บชื่อของตัวแปรในส่วน if ของกฎทุกกฎโดยมีการ define ตัวแปรในภาษา C ดังนี้

```
char *vartab[ MAX_TABENTRY];
```

ซึ่งจะเป็นการ define อะเรย์ แบบ Dynamic โดยที่ค่าของ MAX_TABENTRY จะมีการกำหนดไว้ใน file “define .h” และเมื่อโปรแกรมเริ่มทำงานก็จะมีการ Allocate เนื้อที่ของตัวแปรนี้โดยใช้คำสั่ง

```
for ( I=0 ; I< MAX_TABENTRY; I++)
    vartab[I] = malloc(10);
```

โดยคำสั่งนี้จะเป็นการจองพื้นที่ในหน่วยความจำ เป็น อะเรย์สองมิติ จำนวนเท่ากับ MAX_TABENTRY โดยที่ค่า 10 คือ ค่าความยาวสูงสุดของตัวแปรในกฎใดๆก็ตาม ที่จะสามารถใช้ได้

2. valuetab เป็นตัวแปรอักขระ ทำหน้าที่เก็บค่าของตัวแปร โดยที่ valuetab[i] จะเก็บค่าของตัวแปรที่มีชื่ออยู่ใน vartab[i] ตัวอย่างเช่น

ถ้าเรามีค่าของ valuetab เป็น

vartab[0] = “DOLLAR”

vartab[1] = “FEDINT”

vartab[2] = “FEDMON”

vartab[3] = “INTEREST”

vartab[4] = “STOCKS”

และมีค่าของ valuetab เป็น

valuetab[0] = “rise”

valuetab[1] = “fall”

valuetab[2] = “add”

valuetab[3] = “fall”

valuetab[4] = “rise”

หมายความว่า ตัวแปร STOCK มีค่าเป็น “rise” ตัวแปร FEDMON มีค่าเป็น “fall” ตัวแปร FEDINT มีค่าเป็น “add” ตัวแปร INTEREST มีค่าเป็น “fall” และตัวแปร STOCK มีค่าเป็น “rise” ตามลำดับ

ในการ define ตัวแปร valuetab ในภาษา C มีดังนี้

```
char *valuetab[ MAX_TABENTRY];
```

ซึ่งจะเป็นการ define อะเรย์ แบบ Dynamic โดยที่ค่าของ MAX_TABENTRY จะมีการกำหนดไว้ใน file “define .h” และเมื่อโปรแกรมเริ่มทำงานก็จะมีการ Allocate เนื้อที่ของตัวแปรนี้โดยใช้คำสั่ง

```
for ( I=0 ; I< MAX_TABENTRY; I++)
        valuetab[I] = malloc(10);
```

โดยคำสั่งนี้จะเป็นการจองพื้นที่ในหน่วยความจำ เป็น อะเรย์สองมิติจำนวนเท่ากับ ค่าของตัวแปร MAX_TABENTRY โดยที่ค่า 10 คือ ค่าความยาวสูงสุดของค่าของตัวแปรในกฎใดๆก็ตาม ที่จะสามารถใช้ได้

3. used_list เป็น array ชนิดตัวเลขทำหน้าที่เก็บค่าของกฎที่ถูกใช้ไปแล้วในการอนุมานรอบนั้นๆ โดยในการอนุมานจะทำการตรวจสอบใน used_list ก่อนว่ากฎที่จะนำมาใช้นั้นได้ถูกใช้ไปแล้วในการอนุมานรอบนั้นหรือยัง ถ้าใช้แล้วจะไม่นำกฎนั้นมาใช้อีกเพื่อป้องกันการวนซ้ำของการอนุมาน เมื่อจบการอนุมานในแต่ละรอบก็จะทำการลบค่าข้อมูลใน used_list ออกทั้งหมด

ส่วนติดต่อและควบคุมการทำงานของ XServer

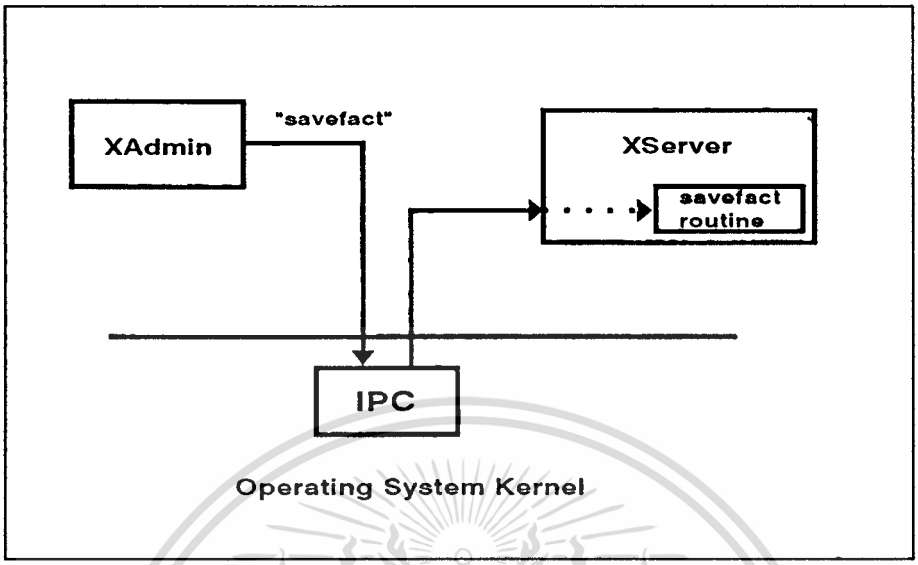
ส่วนนี้จะเป็นส่วนที่ทำการติดต่อกับ XServer เพื่อสั่งให้ XServer ทำงานตามที่ผู้ใช้ต้องการ XAdmin จะสามารถติดต่อกับ XServer ได้โดยผ่านทาง Message queue ซึ่งทางด้าน XServer ได้ทำการสร้างไว้ล่วงหน้าแล้ว ทางด้าน XAdmin จะต้องทำการติดต่อกับ Message queue โดยใช้คำสั่ง "msgget" ดังนี้

```
id=msgget(MKEY1, 0);
```

โดยที่ MKEY1 คือชื่อของ Message queue ซึ่งต้องเป็นชื่อเดียวกับที่ XServer สร้างโดยใช้คำสั่ง "msgget"จากนั้นจึงจะสามารถทำการใช้งาน Message queue นั้นๆได้

การสั่งให้ XServer ทำงานนี้ทำได้โดยการส่งคำสั่งที่ต้องการจาก XAdmin ไปยัง Message queue ที่ XServer รอรับข้อมูลอยู่ในกรณีที่ XServer ได้เริ่มทำงานแล้ว แต่ถ้า XServer ยังไม่เริ่มทำงานเราสามารถสั่งให้ XServer เริ่มทำงานได้โดยใช้คำสั่ง "startsrv" ดังจะได้กล่าวในตอนถัดไป

ข้อมูลที่ส่งไปให้กับ XServer นี้จะเป็นลักษณะของข้อมูลแบบ string เป็นชื่อของคำสั่งที่ต้องการไปให้ XServer สมมติว่าเราต้องการให้ XServer ทำคำสั่ง "savefact" เราสามารถแสดงการทำงานได้ดังรูปที่ 15.



รูปที่ 15.แสดงการรับคำสั่งของ XServer จาก XAdmin

คำสั่งที่สามารถสั่งงานให้กับ XServer ได้มีดังนี้

startsrv เป็นคำสั่งที่สั่งให้ XServer เริ่มการทำงาน โดยโปรแกรม XAdmin จะทำการสั่งให้ระบบสร้าง

โปรเซส XServer ขึ้นมาโดยจะทำงานเป็นโปรเซสที่อยู่เบื้องหลัง ผลของคำสั่งนี้จะเหมือนกับการใช้คำสั่ง "nohup XServer &" ที่พรอมพ์ของยูนิกซ์

stopsrv เป็นคำสั่งให้ XServer หยุดการทำงาน โดยโปรแกรม XAdmin จะทำการส่งข้อมูลเป็นข้อความว่า "exit" ไปยังส่วนรับข้อมูลของ XServer ซึ่งเป็น Message queue สำหรับรอรับข้อมูล เมื่อ XServer อ่านข้อมูลมาจาก Message queue ก็จะแปลความว่าได้มีการสั่งให้ XServer หยุดการทำงาน XServer ก็จะออกจากการทำงานของโปรแกรมและโปรเซสก็จะหยุดทำงาน

xping เป็นคำสั่งที่ใช้ตรวจสอบว่า XServer ได้ถูกเรียกให้ทำงานแล้วหรือไม่ในกรณีที่ XServer ทำงานอยู่ก็จะได้รับข้อความ "XServer is alive" กลับมาจาก XServer และแสดงขึ้นทางหน้าจอ แต่ถ้า XServer ยังไม่ได้ถูกเรียกใช้งาน จะได้รับข้อความ "Cannot get Message queue" แสดงขึ้นทางหน้าจอภาพ ที่ได้รับ ข้อความนี้เป็นเพราะว่า XServer จะเป็นผู้สร้าง Message queue นี้สำหรับรับส่งข้อมูลขึ้นมา ถ้า XServer ยังไม่ได้ถูกเรียกใช้งาน ทำให้ยังไม่มี Message queue นี้เกิด

ขึ้น เมื่อ XAdmin พยายามที่จะติดต่อกับ Message queue นี้เพื่อส่งข้อมูล (ping) จะได้รับการปฏิเสธที่จะติดต่อกับ Message queue นี้จากระบบปฏิบัติการ (Operating System)

savefact เป็นคำสั่งที่สั่งให้ XServer เก็บค่าความจริงซึ่งก็คือค่าของ ตัวแปรต่างๆในระบบลงเก็บในแฟ้มข้อมูลชื่อ fact.base เพื่อไว้ใช้งาน เพราะในกรณีที่มีการ เริ่มการทำงานใหม่ของ XServer ทุกครั้ง ค่าตัวแปรใน XServer จะถูกกำหนดค่าเริ่มต้นใหม่เสมอ การใช้คำสั่ง savefact นี้จะทำให้เราสามารถเก็บค่าสุดท้ายของตัวแปรต่างๆใน XServer ได้ เมื่อเราพิมพ์ข้อมูลในแฟ้มข้อมูล fact.base จะพบว่ามีข้อมูลดังนี้คือ

 # cat fact.base

DOLLAR=rise
FEDMON=null
FEDINT=null
INTEREST=fall
STOCKS=rise

loadfact เป็นการสั่งให้ XServer นำค่าของตัวแปรต่างๆที่เก็บไว้มาใส่ให้กับตัวแปรใน XServer โดยจะแทนที่ค่าที่มีอยู่เดิม

บทที่ 5
ส่วน Xserver

ในบทนี้จะเป็นการกล่าวครอบคลุม ถึงส่วนประกอบและการทำงานของส่วนหลักของระบบที่เรียกว่า Expert System Shell Server หรือ XSever ในส่วนของ XServer จะมีส่วนประกอบต่างๆคือ ส่วนแปลความหมาย, ส่วนอนุมาน, ส่วนติดต่อกับอุปกรณ์รับส่งข้อมูล, และส่วนที่ใช้ในการติดต่อกับส่วนฐานความรู้ ซึ่งรายละเอียดจะมีดังเนื้อหาในหัวข้อถัดไป

ส่วน แปลความหมาย (Interpreter)

ส่วนแปลความหมาย เป็นส่วนที่ทำหน้าที่ในการตรวจสอบและแปลความหมายของการขอใช้บริการ (Request) จาก XClient หรือ XAdmin แล้วส่งให้กับส่วนอนุมานเพื่อทำการหาข้อสรุปของปัญหา เมื่อได้ข้อสรุปของปัญหาแล้วส่วนอนุมานจะส่งกลับมาให้ส่วนแปลความหมายเพื่อทำการแปลว่าจะต้องทำอย่างไรกับผลลัพธ์ที่ได้ และเนื่องจากเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูลนี้ ผู้ใช้สามารถที่จะสร้างโปรแกรมของตนเองขึ้นมา และสามารถเรียกใช้งานโปรแกรมนั้นๆถ้าเงื่อนไขที่กำหนดไว้เป็นจริงได้ ดังนั้นส่วนแปลความหมายนี้จะต้องทำงานร่วมกับส่วนอนุมานกล่าวคือถ้าเงื่อนไขใดที่เป็นจริง (เงื่อนไขในส่วน IF เป็นจริง) ส่วนแปลความหมายจะตรวจสอบส่วนที่เป็นข้อสรุป (ส่วนที่อยู่หลัง THEN) ว่ามีการเรียกใช้งานโปรแกรมอื่นหรือไม่ ถ้าเป็นการเรียกใช้งานโปรแกรมอื่น ส่วนแปลความหมายก็จะเรียกโปรแกรมนั้นขึ้นมาทำงานต่อไป

ส่วนอนุมาน (Inference Engine)

เครื่องอนุมานมีหน้าที่ควบคุมกลไกการอนุมานความรู้ของระบบผู้เชี่ยวชาญ เครื่องอนุมานเป็นส่วนประกอบหลักที่สำคัญส่วนหนึ่งของระบบผู้เชี่ยวชาญซึ่งเป็นตัวกำหนดความสามารถและประสิทธิภาพของระบบผู้เชี่ยวชาญนั้นๆ ภายในเครื่องอนุมานจะมีส่วนที่ทำหน้าที่หาข้อสรุป (Conclusion) ของปัญหาโดยใช้กฎและข้อเท็จจริงในสถานะการณ์นั้นมาใช้ในการสรุปคำตอบซึ่งจากการทำงานของเครื่องอนุมานนี้เราอาจจะเรียกเครื่องอนุมานได้อีกอย่างหนึ่งว่า "เครื่องประมวลผลความรู้" (Knowledge Processing)

วิธีที่นิยมใช้มากที่สุดวิธีหนึ่งคือ Modus Ponens ซึ่งมีหลักการคือ ถ้าส่วนเงื่อนไขของกฎถูกต้องแล้ว ข้อสรุปก็จะถูกต้องด้วยซึ่งถ้า A เป็นส่วนเงื่อนไขและ B เป็นข้อสรุปจะได้ว่าถ้า A ถูกต้องแล้ว B ก็จะถูกต้องด้วยเราสามารถแทนความสัมพันธ์ระหว่าง A และ B ได้ด้วยเครื่องหมาย A -> B

ข้อสังเกต: ถ้า B ถูกต้องแล้ว A ไม่จำเป็นต้องถูกเสมอไป เช่น ประโยค "ถ้าสัตว์ออกลูกเป็นตัวแล้ว สัตว์ชนิดนั้นเป็นสัตว์เลี้ยงลูกด้วยนม" พบว่า ถ้า X เป็นสัตว์เลี้ยงลูกด้วยนมแล้ว X อาจจะไม่ได้ออกลูกเป็นตัวก็ได้ (ตุ่นปากเป็ดเลี้ยงลูกด้วยนมแต่ออกลูกเป็นไข่)

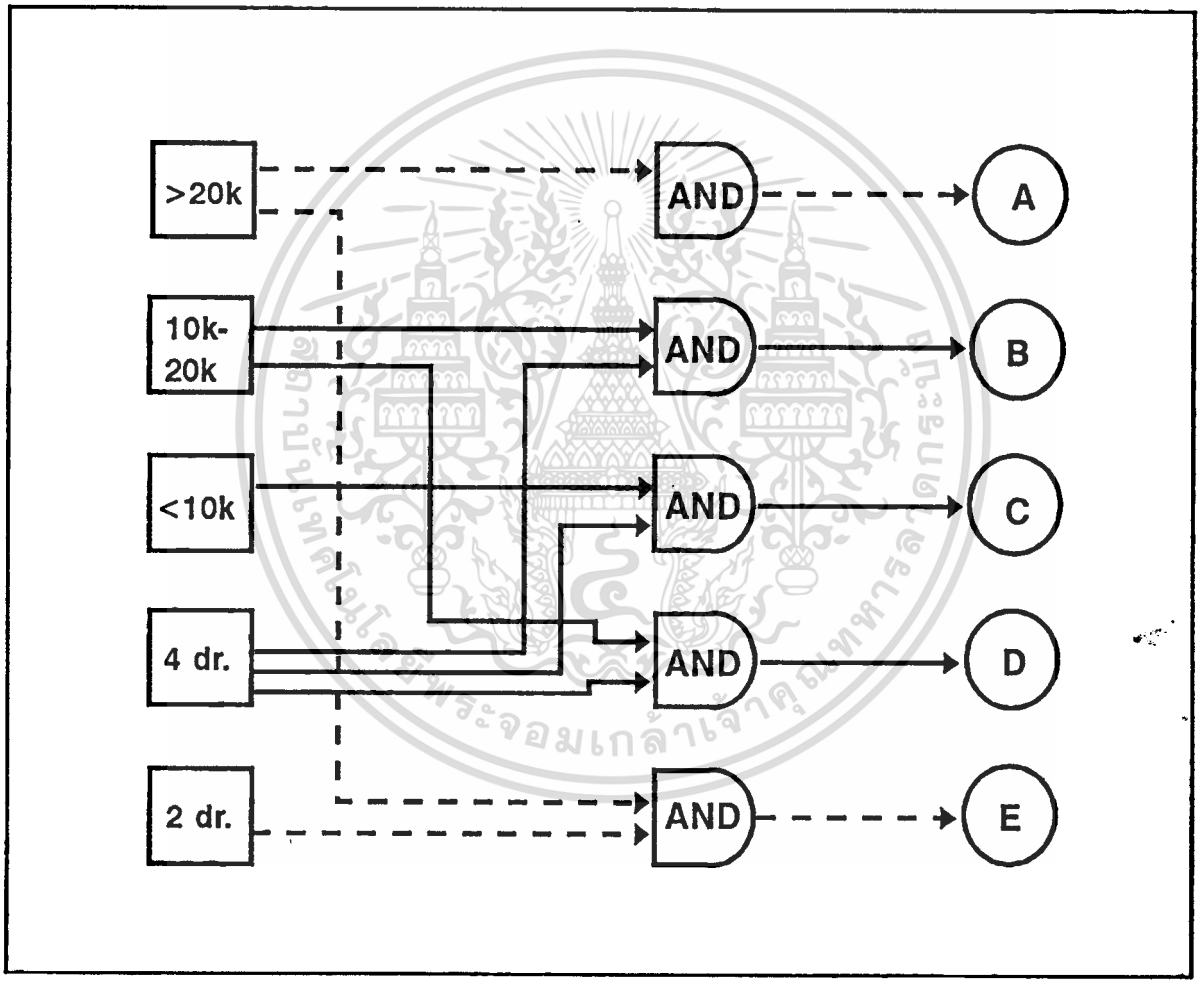
การทำงานของเครื่องอนุมานนี้จะต้องเป็นตัวตัดสินใจในเรื่อง

- จะเริ่มทำการอนุมานอย่างไร
- จะเลือกใช้กฎข้อไหนก่อนถ้ามีหลายกฎที่มีเงื่อนไขถูกต้อง
- การค้นหาจะใช้วิธีใด

ภายในเครื่องอนุมานจะมีฐานความรู้ที่ประกอบด้วยกฎและข้อเท็จจริงต่าง ๆ เหมือนกับฐานความรู้ (Knowledge base) ซึ่งภายในฐานความรู้จะประกอบด้วยกฎที่รวบรวมจากผู้เชี่ยวชาญในเรื่องเฉพาะทาง แต่ในเครื่องอนุมานนี้ประกอบด้วยกฎที่ใช้ในการควบคุมการค้นหาและการสรุปคำตอบสำหรับหาข้อยุติ (Conclusion) โดยที่กฎและข้อเท็จจริงทั้งสองนี้จะถูกเก็บไว้ในส่วนที่แยกกัน

1. การค้นหา จุดมุ่งหมายของระบบผู้เชี่ยวชาญคือการหาและเสนอแนะในการวิเคราะห์หาคำตอบ (หรือกลุ่มของคำตอบของปัญหาที่ให้มา การทำหน้าที่ดังกล่าว ระบบผู้เชี่ยวชาญจะต้องมีการค้นหาข้อมูลสำหรับการแก้ปัญหานั้น ๆ ซึ่งจะอยู่ในส่วนการทำงานของเครื่องอนุมาน ซึ่งเครื่องอนุมานที่ดีจะต้องสามารถวินิจฉัยได้อย่างถูกต้องและรวดเร็ว

ในการค้นหาจะมีรูปแบบใหญ่ ๆ อยู่ 2 แบบ คือแบบเดินหน้า (forward chaining) และแบบย้อนหลัง (backward chaining) ตัวอย่างของการอนุมานทั้งสองแบบมีดังรูปที่ 16.



รูปที่ 16. แสดงการตัวอย่างของการอนุมานแบบเดินหน้าและย้อนกลับ

รูปที่ 16. เป็นการเลือกซื้อรถยนต์โดยทางด้านซ้ายมือ เป็นข้อมูลที่ใช้ในการพิจารณาเลือกส่วนทางด้านขวามือเป็นชนิดของรถยนต์

การอนุมานแบบเดินหน้าเป็นการค้นหาคำตอบจากข้อมูลที่มีอยู่ จากรูปจะเป็นการหาข้อสรุปจากทางด้านซ้ายมือเพื่อเลือกแบบรถยนต์ทางซ้ายมือ เช่น ถ้างบประมาณมีน้อยกว่า 10,000 ดอลลาร์ และต้องการรถแบบ 4 ประตูจะต้องเลือกรถยนต์แบบ C เป็นต้น ส่วนการอนุมานแบบย้อนกลับเป็นการเลือกแบบรถยนต์ทางด้านขวามือแล้วนำมาหาข้อมูลทางด้านซ้ายมือว่าเป็นแบบที่ต้องการหรือไม่ เช่น ถ้าต้องการซื้อรถแบบ B เราก็จะมาดูทางด้านซ้ายมือว่างบประมาณที่ต้องใช้เป็นเท่าไรและมีกี่ประตูถ้าตรงกับความต้องการก็จะเลือกรถชนิดนั้น ถ้าไม่ตรงก็จะนำแบบต่อไปมาพิจารณาในแบบเดียวกัน

ในวิทยานิพนธ์ฉบับนี้ใช้วิธีการอนุมานแบบเดินหน้าในการค้นหาข้อสรุปของปัญหา เพราะหมาะกับการใช้งานในการแก้ปัญหาจากข้อมูลจำนวนน้อยไปหาข้อสรุปที่เป็นไปได้จำนวนมาก การสร้างสามารถทำได้ง่ายไม่ซับซ้อนและการแปลงความรู้เป็นกฎต่าง ๆ สามารถทำได้โดยง่าย สำหรับวิธีการทำงานของการอนุมาน และโครงสร้างข้อมูลจะอธิบายในหัวข้อ 2. และ 3. ตามลำดับ

2. โครงสร้างข้อมูลที่ใช้ในเครื่องอนุมาน โครงสร้างข้อมูลที่สำคัญที่ใช้ในระบบประกอบด้วย

2.1. ฐานความรู้ ทำหน้าที่เก็บฐานความรู้หรือกฎต่าง ๆ ที่ผู้เชี่ยวชาญเป็นผู้ใส่เข้าไปโดยที่การเก็บข้อมูลความรู้จะเก็บเป็นแฟ้มข้อมูลของตัวอักษรที่ผ่านกระบวนการของ Lexical Analyzer แล้ว

2.2 Clause variable list เก็บค่าตัวแปรในส่วนของ IF clause ของกฎแต่ละกฎ เช่น กฎ "if (PRICE > 1000) then RESULT = NO จะพบว่าตัวแปรใน IF clause คือ PRICE ซึ่ง จะถูกเก็บไว้ใน Clause variable list

2.3 Conclusion variable queue เป็น Circular queue ที่ทำหน้าที่เก็บตัวแปรที่ต้องการหาข้อสรุปที่จะใช้ในการทำ Forward chaining

2.4 Conclusion variable queue pointer ทำหน้าที่บอกว่าตอนนี้ระบบกำลังประมวลผลตัวแปรอะไรอยู่

3. ขั้นตอนการอนุมาน เราสามารถนำการทำงานของการอนุมานแบบเดินหน้ามาเขียนสรุปเป็นขั้นตอนได้ดังนี้

3.1 ระบบประกอบด้วยหนึ่งหรือหลาย ๆ สภาวะ (Condition)

3.2 สำหรับแต่ละสภาวะ ระบบจะค้นหากฎในฐานความรู้ (Rule base file) เพื่อหากฎที่มีตัวแปรที่อยู่ในส่วน IF โดยในการหากฎนั้นจะเริ่มจากการนำตัวแปรที่จะหาข้อสรุปมาค้นหาใน Cluase variable list ซึ่ง Clause variable list จะถูกเก็บเป็นแฟ้มข้อมูลตามตัวอักษรเมื่อต้องการค้นหาตัวแปรที่เราต้องการหาข้อสรุปว่ามีอยู่ในกฎข้อใดบ้าง จะต้องนำตัวแปรนั้นไปค้นหาในแฟ้มข้อมูลที่ตรงกับตัวอักษรแรกของตัวแปรนั้น ซึ่งภายใน Clause variable list จะบอกตำแหน่งเริ่มต้นของกฎข้อที่มีตัวแปรนั้นอยู่ใน IF clause

3.3.สำหรับกฎแต่ละกฎสามารถที่จะสร้างสภาวะใหม่ได้จากข้อสรุปที่ได้จากส่วน THEN ซึ่งสภาวะที่ได้ใหม่นี้จะถูกเก็บไว้ใน Conclusion variable queue เพื่อหาผลที่จะเกิดขึ้นจากสภาวะใหม่

3.4. สภาวะทุกสภาวะที่เพิ่มขึ้นมาใหม่ในระบบจะถูกนำไปประมวลผลโดยจะกลับไปทำที่ 3.2. จนกว่าสภาวะใน Conclusion variable queue หมดจึงจบการค้นหา

สำหรับ Flowchart ของเครื่องอนุมานมีอยู่ในภาคผนวก ก.

เพื่อให้เข้าใจในการทำงานของการอนุมานแบบเดินหน้ายิ่งขึ้นจะขอยกตัวอย่างการทำงานแบบ Forward chaining ซึ่งได้ดัดแปลงมาจากตัวอย่างในหนังสือ "A Comprehensive Guide to AI and Expert Systems Using Turbo Pascal" ของ Robert I. Levine, Diane E. Drang ดังนี้

ตัวอย่าง

ถ้าเราต้องการหาว่า ถ้าค่าเงิน Dollar ตกจะเกิดอะไรขึ้นและเรามี กฎดังนี้

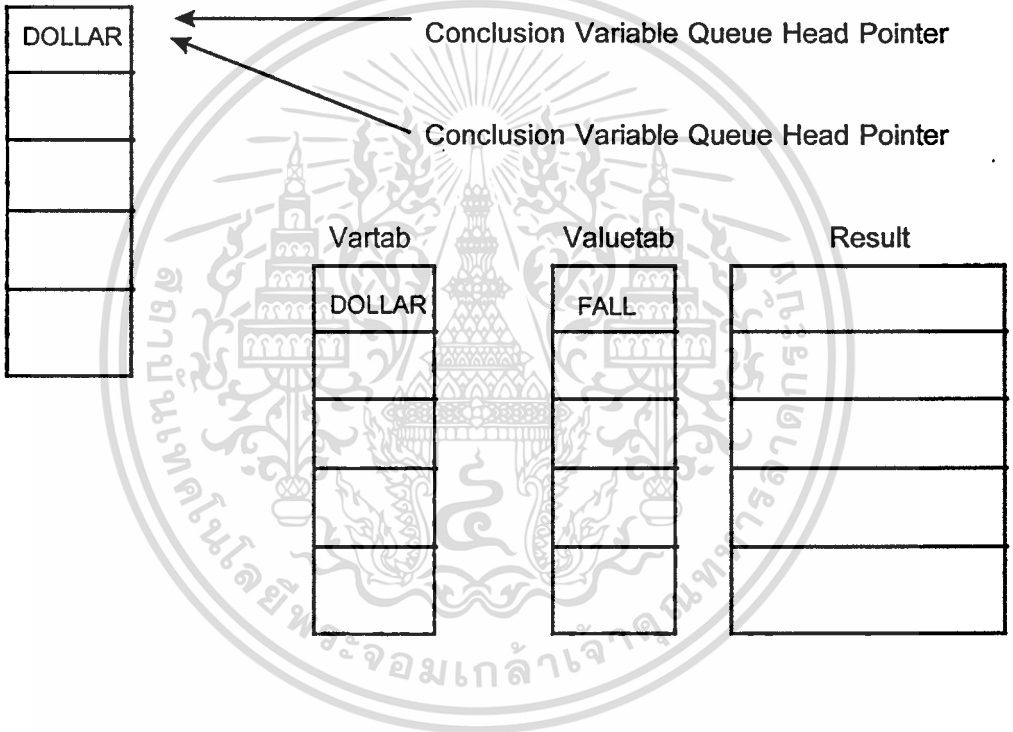
```
Rule 1: if      INTEREST    = FALL
        then    STOCKS      = RISE
Rule 2: if      INTEREST    = RISE
        then    STOCKS      = FALL
Rule 3: if      DOLLAR      = FALL
        then    INTEREST    = RISE
Rule 4: if      DOLLAR      = RISE
```

then INTEREST = FALL

Rule 5: if FEDINT = FALL && FEDMON = ADD

then INTEREST = FALL

จะได้ค่าใน Conclusion variable queue, Vartab, Valuetab และ Result ดังนี้

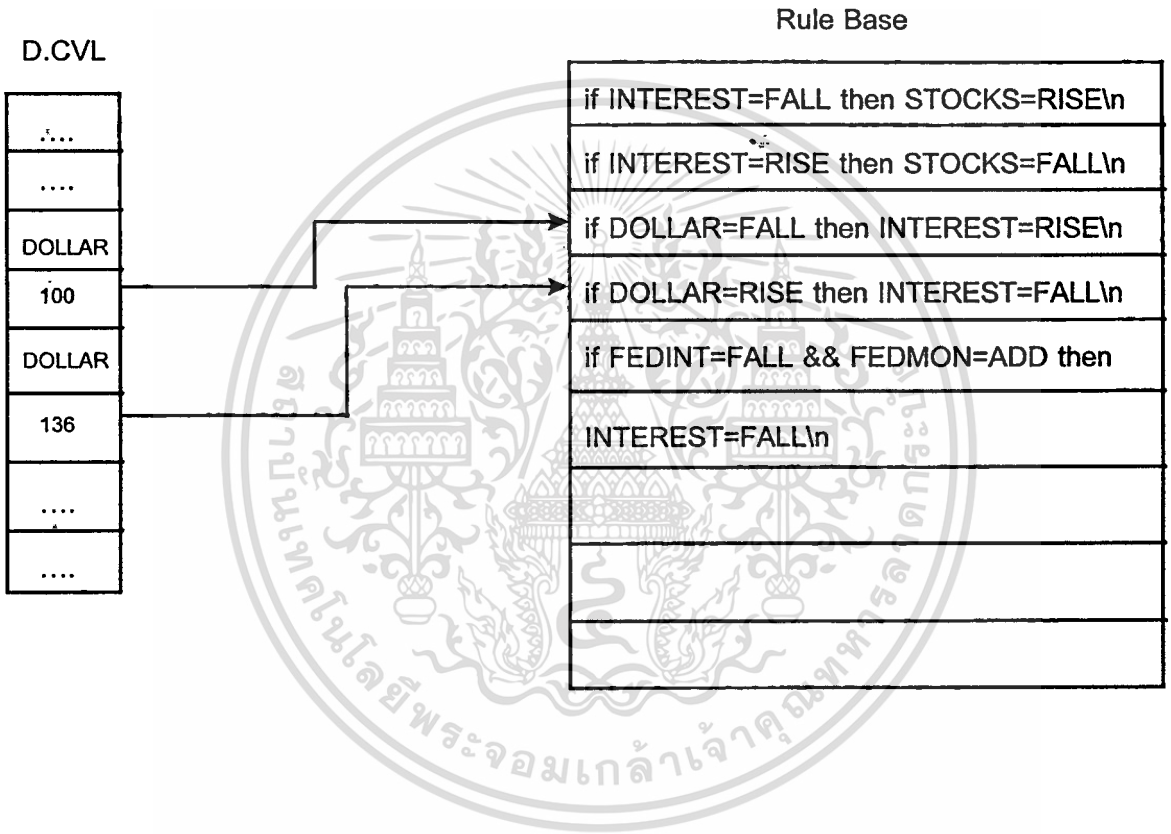


รูปที่ 17.แสดงค่าเริ่มต้นของ Data Structure ต่างๆที่ใช้ในการอนุมาน

หมายเหตุ: Conclusion variable queue เป็น Circular queue แต่ในตัวอย่างนี้จะเขียนในลักษณะของ queue ปกติเพื่อให้ง่ายต่อความเข้าใจ

เมื่อเราต้องการทำ Inference สิ่งที่เราต้องทำสิ่งแรกคือต้องนำตัวแปร "DOLLAR" มาหาว่า กฎข้อใดมีตัวแปรชื่อ "DOLLAR" อยู่ในส่วน IF Clause บ้างโดยเราสามารถหาได้จาก

Clause Variable List File (.cvl) ชื่อ D.cvl ซึ่งได้สร้างขึ้นโดยส่วน Knowledge Acquisition (ดูบทที่ 4.) โดยในตอนที่ ผู้เชี่ยวชาญได้ทำการใส่กฎนั้นๆเข้าไปในระบบ ตัวโปรแกรมจะทำการสร้างแฟ้มข้อมูลนี้ขึ้นมา ซึ่งในแฟ้มข้อมูล D.cvl จะมีข้อมูลดังรูปที่ 18.



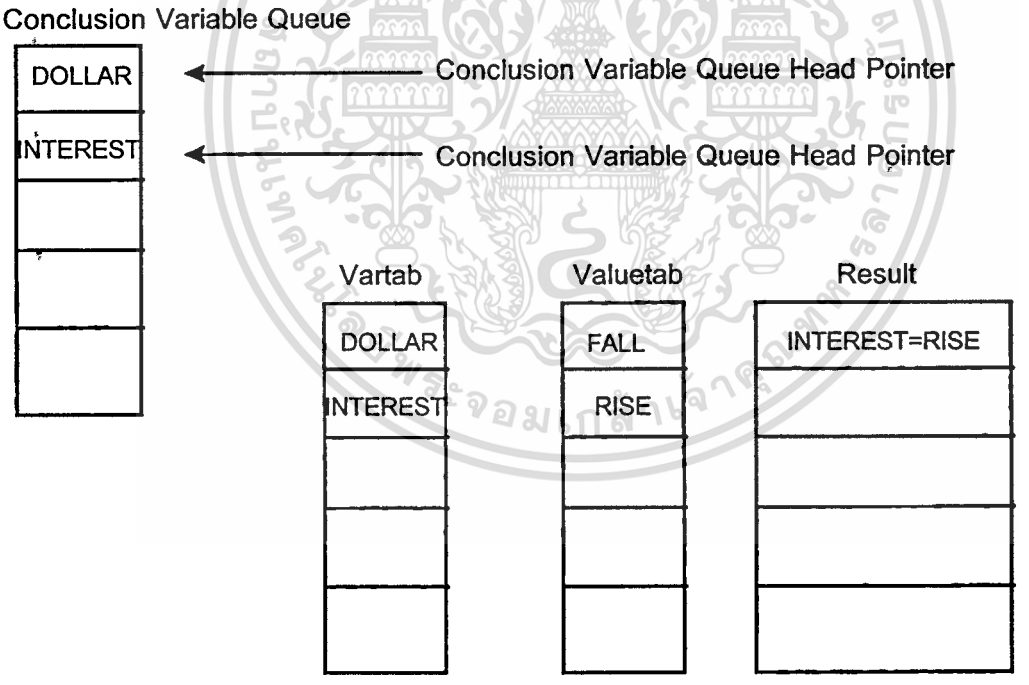
รูปที่ 18.แสดงข้อมูลในแฟ้มข้อมูล D.cvl

จากรูปที่ 18. ค่า "100" จะเป็นค่าของตำแหน่งว่าไบท์ที่ 100 ใน Rule Base จะมีกฎที่มีตัวแปร "DOLLAR" อยู่ในส่วน IF Clause เมื่อเราได้ข้อมูลดังนี้แล้วเราก็จะสามารถดึงกฎนั้นขึ้นมาใช้ได้ทันที ซึ่งเราจะได้กฎ "if DOLLAR=FALL then INTEREST=RISE" เมื่อเราได้กฎที่ต้องการแล้วกฎนี้จะถูกนำไปพิจารณาโดยส่วน Interpreter ในส่วน Interpreter นี้จะทำการตรวจสอบว่า ค่าของข้อมูลในส่วน IF clause มีค่าเป็น "if DOLLAR=FALL" หรือไม่ซึ่งพบว่าเป็นจริง ดังนั้นเราจะได้ข้อสรุปจากส่วน THEN clause ว่า "INTEREST=RISE"

จากข้อสรุปที่ได้เราจะต้องนำมาใส่ใน Conclusion Variable Queue เพื่อที่จะทำการอนุมานต่อว่าการเปลี่ยนแปลงของตัวแปร INTEREST จะทำให้มีการเปลี่ยนแปลงค่าของตัวแปรอื่นๆอีกหรือไม่ ซึ่งตัวแปร INTEREST จะถูกนำมาพิจารณาหลังจากที่ทำการอนุมานค่าของ DOLLAR เสร็จแล้วและตัวระบบจะทำการเลื่อน Conclusion Variable Queue Tail Pointer ไปยังตำแหน่งถัดไปซึ่งก็คือตำแหน่งของ ตัวแปร INTEREST เพื่อบอกระบบให้ทราบว่า ปลายสุดของ Conclusion Variable Queue อยู่ที่ใด

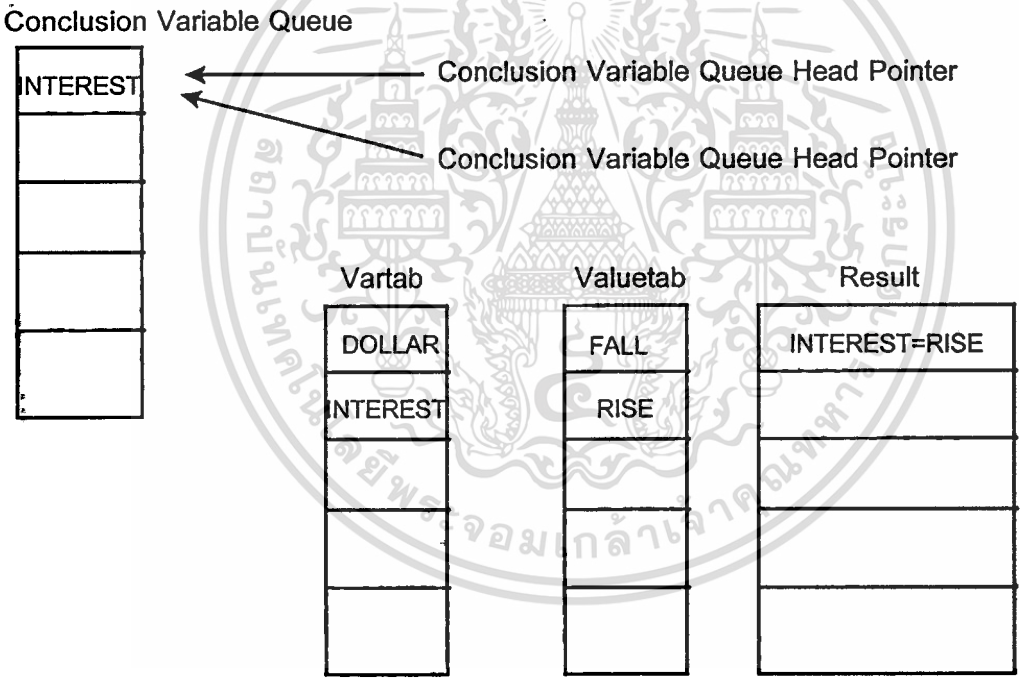
หลังจากนั้นระบบจะนำค่าของ INTEREST ไปใส่ใน Vartab และ Valuetab เพื่อทำการเก็บสถานะของตัวแปรนี้และขั้นสุดท้ายจะนำผลสรุปว่า INTEREST=RISE ไปใส่ใน Result buffer เพื่อทำการเก็บรวบรวมผลสรุปที่ได้ในการพิจารณาในรอบนี้

ค่าของโครงสร้างข้อมูลต่างๆหลังการอนุมานรอบแรกจะเป็นดังรูปที่ 19.



รูปที่ 19. แสดงโครงสร้างข้อมูลหลังการอนุมานครั้งแรก

หลังจากการอนุมานในครั้งแรกแล้วเรายังต้องนำตัวแปร "DOLLAR" มาหาในแฟ้มข้อมูล "D.cvl" อีกว่ายังมีกฎใดที่มีตัวแปร "DOLLAR" อยู่ในส่วน If Clause บ้าง ซึ่งเราพบว่ามีอยู่ในกฎข้อที่ 4. อีก ดังนั้นเราจึงนำกฎข้อนี้มาพิจารณาต่อ แต่เราพบว่า ในกฎข้อนี้ส่วน If Clause มีค่าเป็น "if DOLLAR=RISE" ซึ่งเราสนใจเฉพาะกฎที่มีค่าในส่วน If Clause เป็น "if DOLLAR=FALL" เท่านั้น ดังนั้นเราจึงข้ามกฎข้อนี้ไปและระบบจะค้นหากฎที่ต้องการต่อ แต่พบว่าไม่มีกฎใดอีกที่มีตัวแปร "DOLLAR" อยู่ในส่วน If Clause อีก ระบบจึงจบการอนุมานสำหรับตัวแปร "DOLLAR" เพียงเท่านี้และนำตัวแปร "DOLLAR" ออกจาก Conclusion Variable Queue ดังรูปที่ 20.

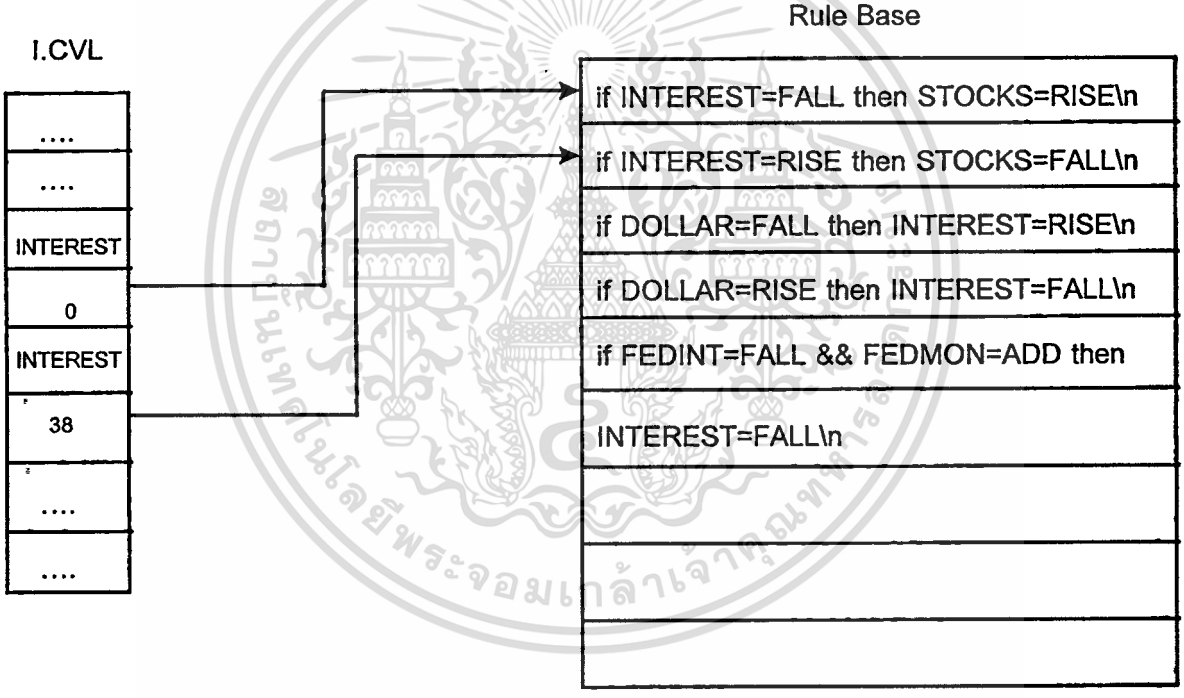


รูปที่ 20.แสดงโครงสร้างข้อมูลของส่วนอนุมานหลังจากจบการอนุมานตัวแปร"DOLLAR"

ถึงแม้ว่าเราจะทำการอนุมานหาผลสรุปจากการเปลี่ยนแปลงตัวแปร "DOLLAR" เสร็จแล้วก็ตาม แต่เรายังไม่เสร็จสิ้นขบวนการอนุมานทั้งหมดเพราะเราต้องหาด้วยว่าผลที่ได้จาก

"DOLLAR=FALL" นั้นเกิดผลอะไรตามมาอีกซึ่งก็คือเราต้องหาต่อว่า เมื่อ "INTEREST=RISE" แล้วจะทำให้ได้ข้อสรุปอะไรเพิ่มขึ้นหรือไม่

จากรูปที่ 20. เราจะพบว่า เมื่อนำตัวแปร "DOLLAR" ออกจาก Conclusion Variable Queue แล้วทำให้ตัวแปรที่อยู่ในส่วนหัวของ Queue คือ "INTEREST" ดังนั้นเราจึงนำตัวแปรนี้มาทำการ อนุมานต่อซึ่งก็เช่นเดียวกับการอนุมานตัวแปร "DOLLAR" ที่ได้ทำมาแล้วข้างต้น เราจะต้องค้นหาว่ามีกฎใดที่มีตัวแปร "INTEREST" อยู่ในส่วน If Clause บ้าง โดยเราสามารถหาได้จากแฟ้มข้อมูลชื่อ "I.cvl" ดังรูปที่ 21.

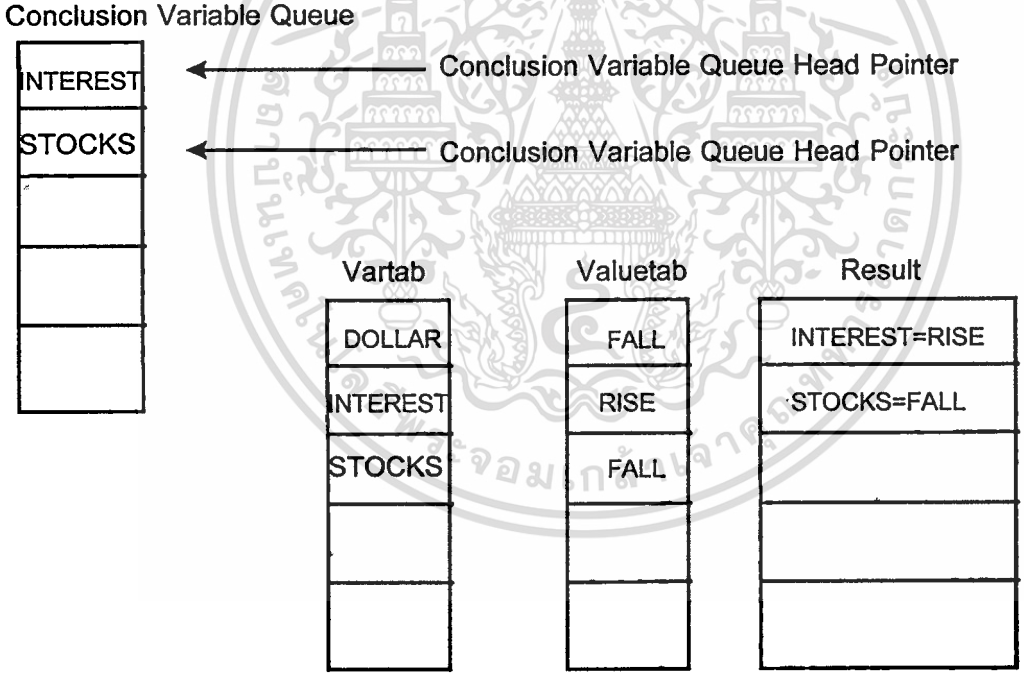


รูปที่ 21. แสดงข้อมูลในแฟ้ม "I.cvl"

จากรูปที่ 21. เราจะได้ว่ามีกฎที่ 1. และกฎที่ 2. ที่มีตัวแปร "INTEREST" ในส่วน If Clause แต่จากกฎข้อที่ 1. เราพบข้อมูลในส่วน If Clause เป็น "if INTEREST=FALL" ซึ่งเราไม่ต้อง

การหาข้อสรุปดังนั้นจึงข้ามกฎข้อนี้ไป ในกฎข้อที่ 2. เราพบว่าข้อมูลในส่วน If Clause มีค่าเป็น "if INTEREST=RISE" ซึ่งเป็นจริงดังนั้นเราจึงนำกฎนี้มาหาข้อสรุปและได้ว่า "STOCKS=FALL"

เมื่อได้ข้อสรุปในการอนุมานครั้งนี้แล้วเราจะต้องนำข้อมูลมา Update ตัวแปรต่างๆของระบบกล่าวคือจะต้องนำตัวแปร "STOCKS" ไปใส่ท้าย Conclusion Variable Queue และเลื่อน Conclusion Variable Tail Pointer ไปชี้ที่ตำแหน่งของตัวแปร "STOCKS" จากนั้นก็จะต้อง Update ตัวแปรอะเรย์ Vartab และ Valuetab เพื่อเก็บสถานะของตัวแปร "STOCKS" และจะต้องนำค่าผลสรุปที่ได้คือ "STOCKS=RISE" ไปใส่ใน Result Buffer ด้วย ดังนั้นสถานะของโครงสร้างข้อมูลต่างๆของส่วนอนุมานจะเป็นดังตัวอย่างในรูปที่ 22.



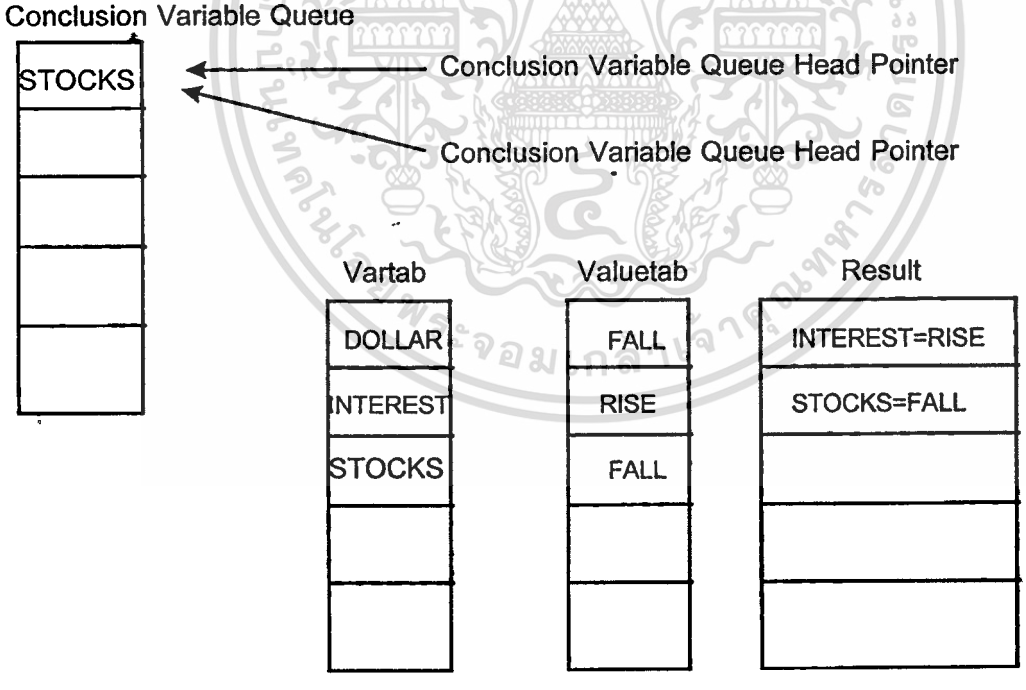
รูปที่ 22.

แสดงโครงสร้างข้อมูลของส่วนอนุมานหลังจากได้ข้อสรุปว่า "STOCKS=FALL"

เมื่อเราตรวจสอบในแฟ้มข้อมูล "I.cvl" แล้วพบว่าไม่มีกฎข้อใดที่มีตัวแปร "INTEREST" อยู่ในส่วน If Clause แล้วดังนั้นเราจึงนำตัวแปร "INTEREST" ออกจาก Conclusion Variable Queue ดังรูปที่ 23. ซึ่งเราจะได้ว่า ตัวแปรที่อยู่ตำแหน่งแรกของ Queue คือ ตัวแปร "STOCKS"

ในขั้นต่อไปเราต้องนำตัวแปรในตำแหน่งแรกของ Conclusion Variable Queue มาทำการอนุมานดังนั้นเราจึงต้องหาว่ามีกฎใดบ้างที่มีตัวแปร "STOCKS" อยู่ในส่วน If Clause ซึ่งเมื่อเราค้นในแฟ้มข้อมูลชื่อ "S.cvl" แล้วจะพบว่าไม่มีกฎข้อใดที่มีตัวแปร "STOCKS" อยู่ในส่วน If Clause อีก ซึ่งจะทำให้เราสามารถสิ้นสุดการอนุมานได้เพียงเท่านี้และได้ข้อสรุปอยู่ใน Result Buffer คือ ถ้า "DOLLAR=FALL" แล้ว จะได้

1. INTEREST=RISE
2. STOCKS=FALL



รูปที่ 23.แสดงโครงสร้างข้อมูลของส่วนอนุมานหลังจากจบการอนุมานตัวแปร "INTEREST"

ส่วนอธิบาย (Explanation Facility)

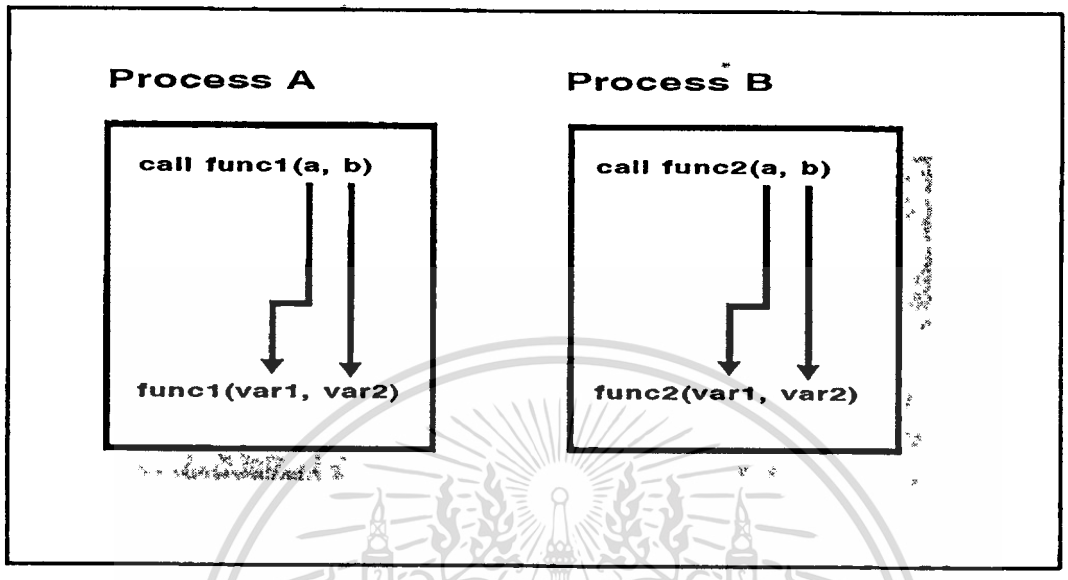
มีหน้าที่ในการเก็บเส้นทาง (Track) ของการอนุมานว่า ได้ใช้กฎไหนบ้างและใช้เหตุผลใดในการสรุปคำตอบ ซึ่งในบางครั้งผู้ใช้ต้องการทราบว่าคำตอบที่ระบบอนุมานนั้นได้มาอย่างไร ผู้ใช้สามารถถามระบบผู้เชี่ยวชาญให้อธิบายที่มาของคำตอบนั้นได้

การทำงานของส่วนอธิบายนี้จะทำโดยการเก็บข้อมูลของการอนุมานในแต่ละขั้นตอนว่าตัวแปรใดมีค่าเปลี่ยนแปลง และเมื่อมีการเปลี่ยนแปลงของตัวแปรจะทำให้สภาวะของระบบเปลี่ยนไป กฎต่างๆจะถูกนำมาใช้เพื่อการหาผลสรุปจากการเปลี่ยนแปลงค่าของตัวแปรนั้นๆ ข้อสรุปของกฎที่หนึ่งอาจจะมีผลให้กฎข้ออื่น ๆ ถูกนำมาใช้ ทำให้เกิดการเปลี่ยนแปลงสภาวะของระบบต่อเนื่องไปจนกว่าการทำงานของส่วนอนุมานจะสิ้นสุดลง ส่วนอธิบายจะต้องเก็บค่ากฎที่ได้ใช้ไปโดยจะเก็บข้อมูลเรียงตามลำดับการถูกเรียกใช้งาน และผลที่ได้จากการทำงานของกฎนั้นโดยเมื่อผู้ใช้งานถามถึงเหตุผลของการอนุมานว่าได้มาอย่างไร ส่วนอธิบายจะต้องนำเอาข้อมูลที่เก็บเอาไว้นี้มาอธิบายให้ผู้ใช้ทราบ

ส่วนติดต่อกับอุปกรณ์รับส่งข้อมูล (I/O Control)

ส่วนรับส่งข้อมูลคือส่วนหนึ่งของระบบซึ่งทำหน้าที่ติดต่อระหว่างระบบกับอุปกรณ์ภายนอกหรือติดต่อกับผู้ใช้ ส่วนนี้เป็นส่วนที่สำคัญที่สุดส่วนหนึ่งของงานวิจัยนี้และเป็นส่วนหนึ่งที่ทำให้เปลือกระบบผู้เชี่ยวชาญนี้แตกต่างจากเปลือกระบบผู้เชี่ยวชาญอื่นๆ ในหัวข้อต่อไปนี้จะเป็นการกล่าวถึงรายละเอียดของการติดต่อระหว่างโปรเซสและการทำงานของส่วนรับส่งข้อมูลของเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล

1. การติดต่อระหว่างโปรเซส (Interprocess Communication : IPC) เป็นการติดต่อของการทำงานแบบหลายโปรเซส (Multiprocess) ความแตกต่างของการติดต่อแบบนี้กับการติดต่อของการทำงานแบบโปรเซสเดียวคือ ในแต่ละ Module ของการทำงานแบบโปรเซสเดียวจะมีการติดต่อกันโดยใช้ตัวแปร, function call หรือการผ่าน argument ที่ใช้ใน function call นั้นๆดังรูปที่ 24.

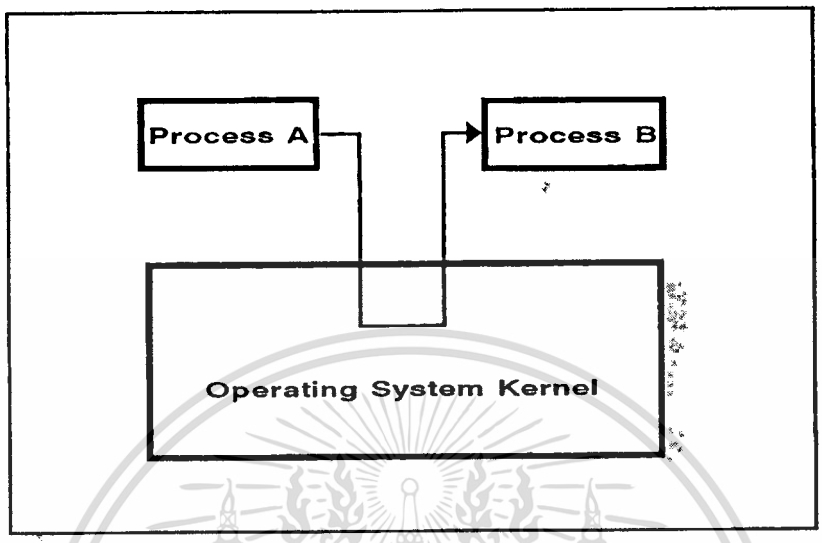


รูปที่ 24. แสดงการส่งผ่านข้อมูลในโปรเซสเดียวกัน

การส่งผ่านข้อมูลในโปรเซสหรือในโปรแกรมเดียวกันนี้สามารถทำได้ง่ายเพราะว่า ในโปรเซสเดียวกันนั้นสามารถอ้างตัวแปรซึ่งเป็นการอ้างตำแหน่ง (address) ในหน่วยความจำของตัวโปรเซสนั้นๆเองเพื่อเขียนหรืออ่านข้อมูลได้

แต่ในการส่งผ่านข้อมูลระหว่างโปรเซสจะมีความซับซ้อนกว่า เพราะการอ้างตำแหน่งเพื่ออ่านหรือเขียนข้อมูลในหน่วยความจำของโปรเซสอื่นในระบบปฏิบัติการแบบหลายโปรเซสนั้นทำได้ยากเนื่องจากเราไม่สามารถทราบได้ว่าในขณะใดขณะหนึ่งข้อมูลที่เราต้องการอ่านหรือเขียนอยู่ในหน่วยความจำ (Physical address) ตำแหน่งใดนอกจากนี้การอ่านหรือเขียนข้อมูลผิดที่อาจทำให้การทำงานของระบบผิดพลาดได้

การติดต่อระหว่างโปรเซส 2 โปรเซสในระบบยูนิกซ์ทำได้โดยผ่านทางแก่น (kernel) ของระบบปฏิบัติการ (Operating System : OS) ดังรูปที่ 25.

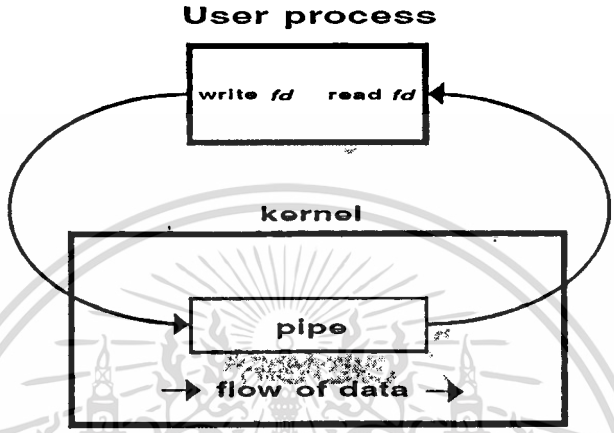


รูปที่ 25. แสดงการติดต่อระหว่างโปรเซสโดยผ่านทางแก่นของระบบ

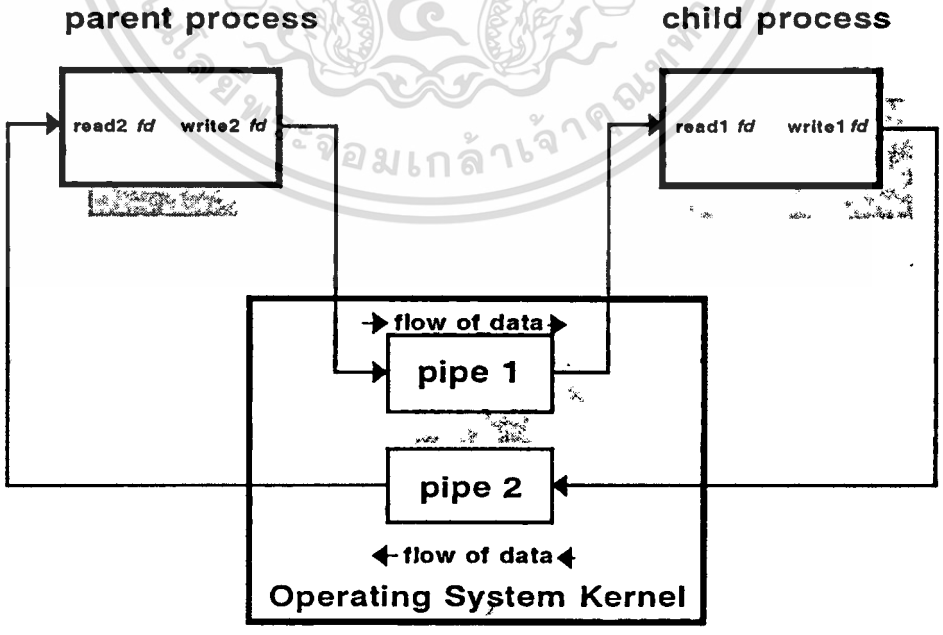
2 .ชนิดของ IPC

ในระบบ UNIX มีรูปแบบของการติดต่อระหว่างโปรเซสหลายอย่างคือ

2.1 ไพพ์ (Pipe) ไพพ์เป็นลักษณะคล้ายท่อส่งข้อมูลแบบ FIFO (First In First Out) โดยจะสามารถส่งข้อมูลได้ทิศทางเดียวผ่านทาง kernel ของระบบ ดังรูปที่ 26. ซึ่งถ้าต้องการให้สามารถส่งข้อมูลได้สองทิศทางจะต้องใช้ pipe สองชุด ชุดหนึ่งสำหรับรับข้อมูล อีกชุดสำหรับส่งข้อมูลดังรูปที่ 27. เราสามารถสร้างไพพ์ได้โดยใช้ "pipe" system call



รูปที่ 26. แสดงการรับส่งข้อมูลทิศทางเดียวโดยผ่านทางไพพ์



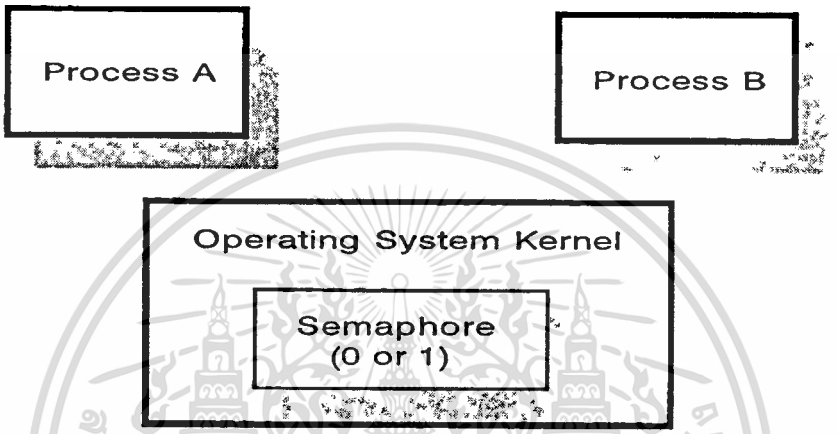
รูปที่ 27. แสดงการส่งข้อมูลระหว่างโปรเซสแบบทิศทางเดียวโดยใช้ pipe 2 ชุด

ข้อมูลที่รับส่งโดยใช้ไพพ์ จะเป็นข้อมูลแบบ Stream I/O ซึ่งจะไม่มี record boundaries การรับส่งข้อมูลจะต้องกำหนดขนาดของข้อมูลที่จะอ่านหรือเขียนด้วย เช่นเมื่อต้องการส่งข้อความว่า Helloจะต้องระบุในโปรแกรมด้วยว่าจะส่งข้อมูลขนาด 5 ตัวอักษร ทางฝ่ายรับก็จะต้องรู้ด้วยว่าข้อมูลที่จะอ่านนั้นมีขนาด 5 ตัวอักษร ซึ่งถ้าอ่านเกินจากนี้อาจเป็นการอ่านค่าของ record อื่นติดขึ้นมาด้วย ดังนั้นผู้พัฒนาโปรแกรมที่ใช้ไพพ์นี้จะต้องมีการสร้างข้อกำหนด (protocol) ของโปรแกรมนั้นๆขึ้นมาเพื่อที่จะให้การทำงานของโปรเซสรับและส่งนั้นทำงานสัมพันธ์กัน

2.2 Named Pipe เป็นลักษณะและการทำงานเป็นแบบ FIFO และใช้ส่ง Stream เหมือนไพพ์ธรรมดาแต่แตกต่างกันตรงที่ Named pipe มีชื่อและมีลักษณะเช่นเดียวกับ Device file ซึ่งสามารถสร้างได้โดยคำสั่ง mknod เป็นคำสั่งเดียวกับที่ใช้สร้าง Device file , Named pipe เป็น IPC ของ ยูนิกซ์System V ไม่มีใน 4.3BSD ยูนิกซ์

2.3 Semaphore เป็นลักษณะของตัวแปรเลขจำนวนเต็ม (Integer) ซึ่งถ้ามากกว่า 0 หมายถึงresource นั้นๆว่างอยู่ ถ้าโปรเซส A ต้องการใช้ resource สามารถใช้งานได้และจะทำการเพิ่มค่าของSemaphore อีก 1 โปรเซสอื่นก็จะใช้งานไม่ได้ เมื่อโปรเซส A ทำงานเสร็จก็จะลดค่ากลับมาเป็น0 เหมือนเดิมดังนั้นเมื่อโปรเซสอื่นที่ต้องการใช้งาน resource ก็จะสามารถใช้งานได้ และจะทำการเพิ่มค่าของ Semaphore อีก 1 เหมือนโปรเซส A

เนื่องจาก Semaphore ใช้ในการ Synchronize การทำงานของโปรเซสต่างๆไม่ให้มีการแย่งกันใช้ resource ดังนั้นจะต้องไม่มีการแย่งกันใช้ Semaphore ด้วย เพราะถ้าดูการใช้งาน Semaphoreแล้วจะพบว่ามีการทำงานเป็น 2 คำสั่งคือ อ่านค่าว่าเป็น 0 หรือไม่ และ ถ้าพบว่าไม่เป็น 0 ก็จะทำการลดค่า Semaphore นั้น กล่าวคือ ในขณะที่โปรเซส A กำลังอ่านค่า Semaphore แล้วพบว่ามากกว่า 0 และกำลังจะลดค่าลงมาเป็น 0 เพื่อป้องกันไม่ให้โปรเซสอื่นๆใช้ resource แล้วพบว่าในขณะเดียวกันนี้ก็มีโปรเซส B เข้ามาอ่านค่าของ Semaphore นี้เหมือนกัน โปรเซส B ก็จะพบว่าค่าของ Semaphoreมากกว่า 0 (เพราะโปรเซส A ยังไม่ได้ลดค่าของ Semaphore) ก็จะทำให้การ synchronize ของทั้งสอง โปรเซสผิดพลาดคือ โปรเซส B จะสามารถใช้ resource ได้พร้อมกับโปรเซส A การแก้ไขเหตุการณ์นี้ก็คือ ต้องไม่อนุญาตให้โปรเซสอื่นจะต้องไม่สามารถอ่านค่า Semaphore ในขณะนั้นได้และจะต้องรอให้โปรเซส A ใช้งาน Semaphoreเสร็จก่อน ซึ่งการทำเช่นนี้จำเป็นต้องนำ Semaphore ไปไว้ในส่วน kernel ของ OS ดังรูปที่ 28. เพราะสามารถทำให้คำสั่งที่ใช้งาน Semaphore รวมเป็นคำสั่งเดียว (Atomic operation) ได้และสามารถรับรองได้ว่าการใช้งานของ Semaphore จะไม่ถูกแย่งใช้โดยหลายๆโปรเซส



รูปที่ 28. แสดงการเก็บข้อมูลของ Semaphore ใน kernel

จุดประสงค์ในการใช้งาน Semaphore นั้นไม่ได้ใช้สำหรับรับส่งข้อมูลที่มีขนาดใหญ่ๆ เหมือนไพพ์ แต่เป็นกลไกที่ใช้สำหรับการ Synchronize การทำงาน (อ่านหรือเขียนข้อมูล) ระหว่างโปรเซสที่ต้องการใช้งาน resource ของระบบ แต่โดยทั่วไปจะใช้กับการใช้งาน Shared memory ซึ่งจะได้กล่าวถึงในหัวข้อต่อไป

2.4 Shared memory ในการทำงานแบบ Client-Server สำหรับ copy ข้อมูลจากแฟ้มข้อมูลจะต้องทำดังนี้คือ (รูปที่ 29.)

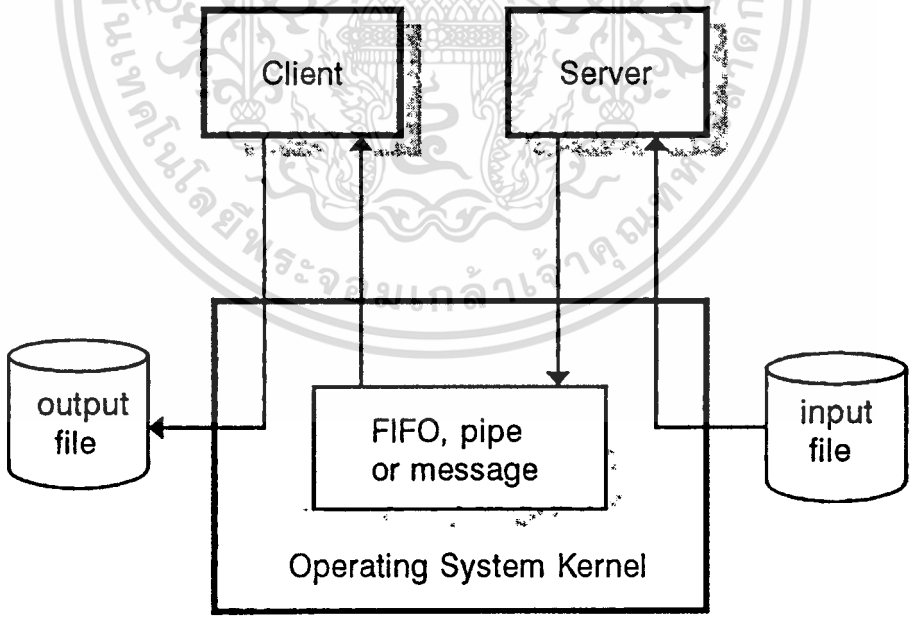
- Kernel อ่านข้อมูลจากแฟ้มข้อมูลต้นทางมาใส่ในเนื้อที่ของ kernel แล้ว ส่งข้อมูลนั้นให้Server
- Server เขียนข้อมูลลงใน pipe หรือ Message queue ซึ่งต้องเขียนลงใน kernel
- Client อ่านค่าข้อมูลจากใน kernel แล้วนำไปเก็บในเนื้อที่ของ client

- Client นำค่าที่อ่านได้เขียนลงในแฟ้มข้อมูลปลายทาง ซึ่งในการเขียนนี้ก็จะต้องเขียนลงในเนื้อที่ของ kernel แล้ว kernel ก็จะส่งไปเขียนที่แฟ้มข้อมูลปลายทาง ซึ่งจะเห็นได้ว่าจะต้องใช้การ copy ข้อมูลโดยผ่านทาง kernel ถึง 4 ครั้ง

Shared memory เป็นการใช้ข้อมูลในหน่วยความจำร่วมกันระหว่างสอง หรือหลายๆ โปรเซส ซึ่งจะต้องใช้ Semaphore เป็นเครื่องช่วยในการจัดการการใช้งาน การใช้ Shared memory นี้จะทำให้จำนวนครั้งของการอ่านเขียนข้อมูลลงใน kernel ลดลง กล่าวคือจะมีการ copy ข้อมูลผ่านทาง kernelเพียง 2 ครั้งเท่านั้น ขั้นตอนในการทำงานจะเป็นดังนี้

- Server ตรวจสอบค่า Semaphore เพื่อที่จะขอใช้ Shared memory ถ้า Semaphore มีค่ามากกว่า 0 Server จะสามารถใช้ Shared memory นั้นได้และจะเปลี่ยนค่านั้นเป็น 0 เพื่อไม่ให้โปรเซสอื่นเข้ามาอ่านเขียนข้อมูลใน Shared memory

- Server อ่านข้อมูลจากแฟ้มข้อมูลต้นทางไปใส่ใน Shared memory

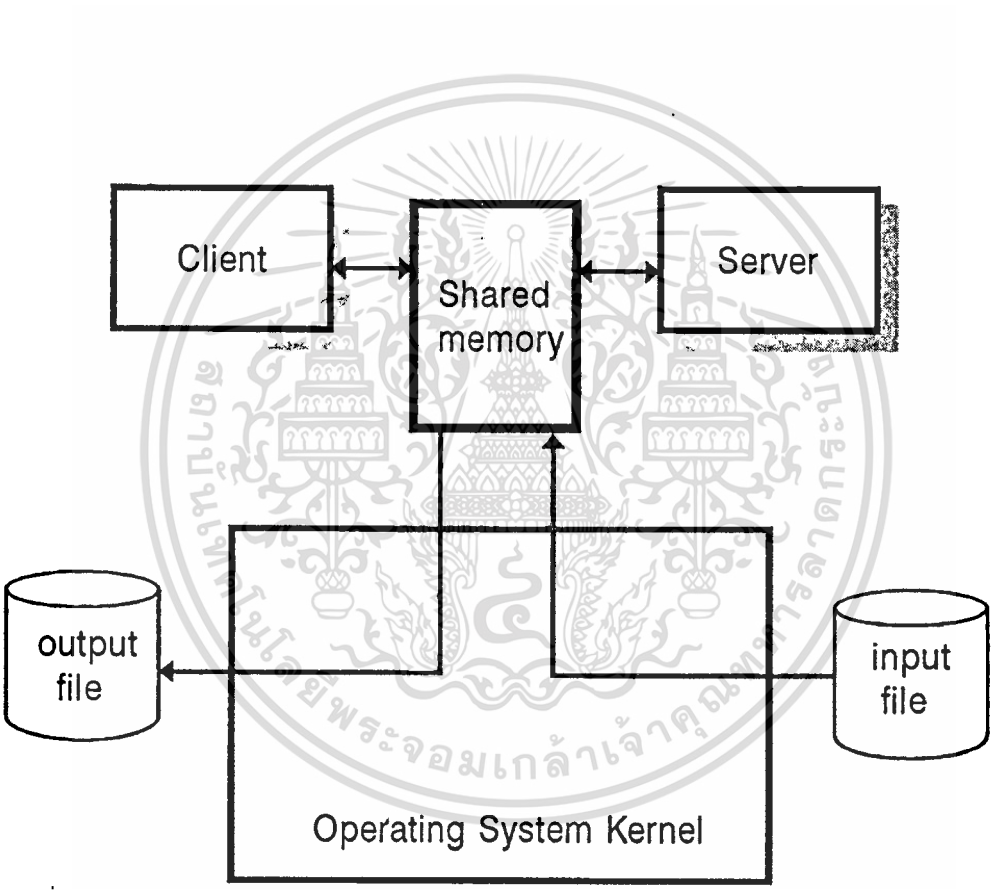


รูปที่ 29.แสดงการ copy ข้อมูลระหว่าง Client และ Server ผ่านทาง pipe หรือ Message queue

- Server แจ้งให้ Client ทราบว่ามีข้อมูลพร้อมที่จะให้อ่านแล้วโดยเปลี่ยนค่า Semaphore ให้มากกว่า 0

- Client นำข้อมูลจาก Shared memory ไปเขียนในแฟ้มข้อมูลปลายทาง

รูปที่ 30.เป็นรูปที่แสดงการอ่านเขียนข้อมูลของ Client และ Server โดยผ่านทาง Shared memory



รูปที่ 30. แสดงการ copy ข้อมูลระหว่าง Client และ Server ผ่านทาง Shared memory

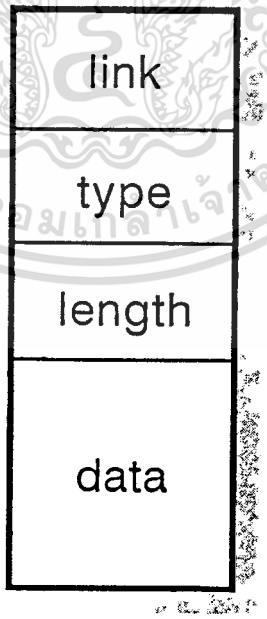
5.4.2.5 Message queue

เป็นการรับส่งข้อมูลในรูปของ Message ซึ่งโครงสร้างข้อมูลจะอยู่บนระดับของ Stream อีกชั้นหนึ่งซึ่งการรับส่งแบบ Message นี้จะดีกว่าแบบ Stream คือ เราสามารถกำหนด record boundary ได้โดยที่ในโครงสร้างข้อมูลแบบ Message จะมีส่วนที่กำหนดขนาดของ record ซึ่งผู้ที่พัฒนาระบบไม่จำเป็นต้องคำนึงถึงเรื่อง record boundary เพราะระบบยูนิกซ์จะเป็นตัวจัดการให้ จากการทดลองใช้งาน IPC ทั้ง 4 อย่างที่ได้กล่าวมานี้พบว่าเราสามารถใช้ IPC ได้ทุกรูปแบบในโครงงานนี้ แต่ Message queue จะเหมาะสมกับการทำงานของโครงงานนี้ที่สุด เพราะผู้พัฒนาโปรแกรมไม่ต้องมีการจัดการเกี่ยวกับ record boundary เองเหมือนกับการใช้งานไพพ์ และมีความง่ายในการใช้งานมากกว่าการใช้ Shared memory และ Semaphore

สำหรับรายละเอียดของการใช้ Message queue ในโครงงานนี้จะได้กล่าวถึงในหัวข้อถัดไป

3. Message queue ในเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูล

3.1. โครงสร้างของ Message queue โครงสร้างของ Message แต่ละ Message ใน Message queue จะมี Attribute ประจำ Message ดังรูปที่ 31.



รูปที่ 31. แสดง โครงสร้างของ Message

สำหรับข้อมูลในโครงสร้างข้อมูลของ Message มีความหมายดังนี้

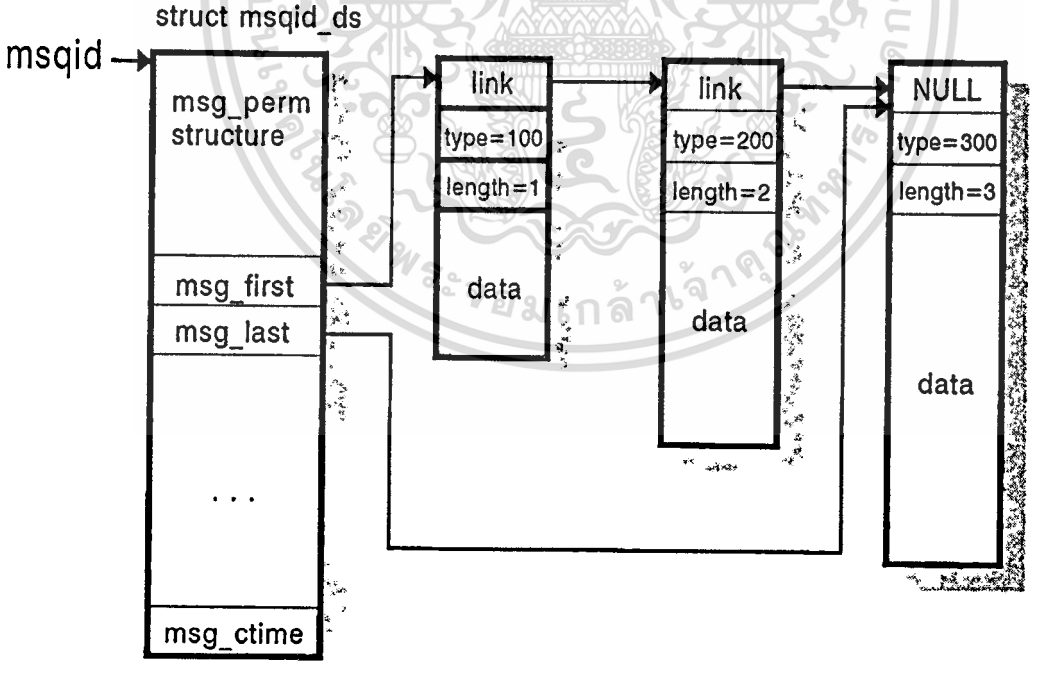
- ลิงค์ (link) เป็น singly link list ชี้ไปที่ Message ถัดไป

- ชนิด (Type) ของ Message เป็น long integer ใช้เป็นตัวกำหนดความแตกต่างของ Message แต่ละ Message เราสามารถกำหนดได้ว่าการรับส่งข้อมูลจะรับส่งข้อมูลที่เป็นชนิดใดได้ เช่นกำหนดให้ผู้รับรับเฉพาะ Message ชนิด 100 เท่านั้นเป็นต้น นอกจากนี้แล้วเรายังสามารถใช้ชนิดของ Message เป็นตัวอ้างถึงตำแหน่ง (Address) ของ โปรเซสแต่ละโปรเซสได้

- ความยาว (length) เป็นข้อมูลที่บอกขนาดของ Message นั้นๆมีหน่วยเป็นไบท์

- ข้อมูล (data) คือเนื้อข้อมูลที่ผู้ใช้ต้องการส่ง

สำหรับในแก่นของระบบยูนิกซ์จะมีโครงสร้างข้อมูลที่ใช้จัดการกับ Message ต่างๆ ใน Message queue ดังรูปที่ 32.



รูปที่ 32. แสดงโครงสร้างของ Message queue ในแก่นของระบบ

ในเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูลนี้มีโปรเซสที่เรียกว่า XClient (eXpert system Client) ทำหน้าที่ในการรับข้อมูลจากอุปกรณ์ที่ต้องการเช่น เครื่องชั่งน้ำหนัก เครื่องตรวจวัดอุณหภูมิ หรือ แป้นพิมพ์แล้วเปลี่ยนรูปแบบ (Format) ของข้อมูลให้เป็นรูปแบบที่ส่วนประมวลผลความรู้ (eXpert Server : XServer) เข้าใจได้แล้วส่งข้อมูลนี้ให้กับส่วนประมวลผลความรู้อีกทีหนึ่งโดยผ่านทาง Message queue ที่เชื่อมต่อกัน ส่วน XClient นี้เป็นส่วนที่ผู้ใช้สามารถเขียนขึ้นเองเพื่อให้เข้ากับอุปกรณ์ที่ต้องการได้

การส่งผ่านข้อมูลระหว่าง XServer และ Xclient นั้นทำได้โดย XClient ส่งข้อมูลผ่านทาง Message queue โดยทาง XServer จะต้องทำการสร้าง Message queue ขึ้นมาก่อนที่ XClient จะติดต่อกับ Message queue เสมอการสร้าง Message queue ทำได้โดยทางฝั่ง XServer ใช้คำสั่ง

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(key_t key, int msgflag);
```

key_t เปรียบได้กับชื่อของ Message queue นั้นซึ่งเป็นข้อมูลชนิด integer 32 bit โดย Message queue แต่ละตัวจะต้องมีค่า key_t ไม่ซ้ำกัน โปรเซสที่ต้องการใช้ Message queue ในการติดต่อกันจะต้องใช้ค่า key_t เดียวกัน msgflag เป็นตัวบอกลักษณะการใช้งานของ Message queue นั้นว่าจะมีการใช้งานได้อย่างไร ค่าของ msgflag ที่เป็นไปได้มีดังนี้

ตารางที่ 1.

Numeric	Symbolic	Description
0400	MSG_R	Read by owner
0200	MSG_W	Write by owner
0040	MSG_R>>3	Read by group
0020	MSG_W>>3	Write by group
0004	MSG_R>>6	Read by world
0002	MSG_W>>6	Write by world
	IPC_CREAT	Create new entry. If an existing entry if found that entry is returned.
	IPC_EXCL	Setting IPC-CREAT and IPC_EXCL bits create a new entry only if the entry does not already exist. If an existing entry is found, an error ccurs.

ตารางที่ 1. แสดงค่าที่เป็นไปได้และความหมายของ msgflag

ในโครงงานนี้จะมีการสร้าง Message queue โดยส่วน XServer โดยใช้พารามิเตอร์ดังนี้

```
#define MKEY1 1234L;

if ((id = msgget(MKEY1, PERMS | IPC_CREATE)) < 0)
        printf("Server: cannot get Message queue\n");
```

โดย id เป็นตัวแปรที่กำหนดให้โปรแกรม XServer รู้จัก Message queue ชื่อ (key_t) 1234 เมื่อ XServer ใช้คำสั่งนี้แล้วก็จะมีการสร้าง Message queue ที่มีชื่อว่า 1234L ขึ้นมาในแก่นของระบบถ้าโปรเซส XClient ใดๆที่ต้องการจะติดต่อกับ Message queue นี้จะต้องใช้คำสั่ง

```
#define MKEY1 1234L;    /* ชื่อของ Message queue     */
#define PERMS 0666;     /* Message queue permission */

if((id = msgget(MKEY1, 0)) < 0)
        printf("Client: cannot get Message queue\n");
```

คำสั่งนี้จะไม่ทำการสร้าง Message queue ขึ้นมาอีกเนื่องจาก msgflag เป็น 0 แต่จะทำการติดต่อกับ Message queue ที่ชื่อว่า 1234L ที่ทาง XServer ได้สร้างไว้แล้วถ้า XServer ยังไม่ได้สร้าง Message queue นี้ทางด้าน XClient จะได้รับคำตอบว่าไม่สามารถใช้ Message queue นั้นได้

หลังจากที่ทั้งสองด้านได้ทำการสร้างและติดต่อกับ Message queue แล้ว ก็จะสามารถทำการส่งรับข้อมูลถึงกันได้โดยใช้คำสั่ง msgsnd และ msgrcv ซึ่งมีรูปแบบการใช้งานดังนี้

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd(int msqid, struct msgbuf *ptr, int length, int flag);
```

ค่าของ msqid ก็คือค่าเดียวกับ id ที่ได้จากคำสั่ง msgget ส่วนค่าของ ptr คือ pointer ที่ชี้ไปยังโครงสร้างข้อมูลดังนี้

```
#include <sys/msg.h>

struct msgbuf { long mtypes;   /* ชนิดของ message    (ดูรูปที่ 29 โครงสร้าง message  */
char mtext[1];  /* ข้อมูลของ message นั้น (ดูรูปที่ 29 โครงสร้าง message */
};
```

-length เป็นค่าที่ต้องใส่เข้าไปให้กับคำสั่งนี้ โดยเป็นค่าความยาวของ ข้อมูล

-flag มีค่าได้ทั้ง IPC_NOWAIT หรือ 0 ถ้าเป็น 0 IPC_NOWAIT หมายถึงว่าเมื่อเรียกใช้คำสั่ง msgsend แล้วไม่ว่าจะส่งสำเร็จหรือไม่จะกลับมาที่โปรแกรมในคำสั่งต่อไปเลยโดยไม่รอการ

ทำงานซึ่งถ้าการส่งสำเร็จก็จะได้ผลลัพธ์ (return value) เป็น 0 ถ้าส่งไม่สำเร็จ เช่น Message queue นั้นเต็ม ก็จะได้ผลลัพธ์เป็น -1 แต่ถ้าเราระบุ flag เป็น 0 ถ้า Message queue เต็มโปรแกรมก็จะรอจนกว่าจะ Message queue นั้นว่างเมื่อส่งได้แล้วก็จะกลับมาทำโปรแกรมในคำสั่งต่อไป

ตัวอย่างการส่งข้อมูลซึ่ง ทั้ง XServer และ XClient จะใช้เหมือนกันมีดังนี้

```
msgsend(id, (char *) &(mesgptr->mesg_type), mesgptr->mesg_len, 0)
```

สำหรับการรับ message นั้นสามารถทำได้โดยคำสั่ง msgrecv ดังนี้

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgrcv(int msqid, struct msgbuf *ptr, int length, long msgtype, int flag);
```

ความหมายของตัวแปร msqid และ msgbuf จะเหมือนกับคำสั่ง msgsend แต่ length, msgtype, flag จะมีความหมายดังนี้

-length หมายถึงขนาดใหญ่ที่สุดของ message ที่สามารถรับได้ซึ่งก็คือขนาดของตัวแปรที่เราจองไว้สำหรับรับ message นั้นนั่นเอง

-msgtype คือชนิดของ message ที่กำหนดให้คำสั่ง msgrcv รับเข้ามาซึ่งจะนำไปเปรียบเทียบกับค่า type ของ message ใน Message queue โดยมีข้อกำหนดดังนี้

- ถ้า msgtype เท่ากับ 0 คำสั่งนี้จะอ่าน message แรกใน Message queue เข้ามาใส่ในตัวแปร msgbuf

- ถ้า msgtype มากกว่า 0 คำสั่งนี้จะอ่าน message แรกที่มี type เดียวกันกับ msgtype เข้ามาใส่ msgbuf

- ถ้า msgtype น้อยกว่า 0 คำสั่งนี้จะอ่านค่า message ที่มีค่า type น้อยกว่าหรือเท่ากับ msgtype เข้ามาใส่ใน msgbuf

- สำหรับค่า flag จะเป็นตัวระบุการทำงานของคำสั่งนี้โดย flag ที่ใช้บ่อยๆคือ

MSG_NOERROR ถ้าใช้ flag ตัวนี้ เมื่อมีการส่ง message ที่มีขนาดใหญ่เกินค่า length คำสั่งนี้จะทำการตัดให้ เท่ากับค่า length และจะไม่ return error ถ้าไม่ระบุจะ return error ว่า ขนาดของ message ใหญ่ไป

IPC_NOWAIT ให้การทำงานของโปรแกรมกลับมาทำงานต่อทันที ในกรณีที่อ่าน message ได้ ก็จะได้ผลลัพธ์เป็น 0 ถ้าอ่านไม่ได้จะได้ผลลัพธ์เป็น -1 ถ้าไม่ใช้ flag ตัวนี้ ในกรณีที่อ่านค่า message ไม่ได้ ตัวอย่างเช่น

ไม่มี message ใน Message queue คำสั่งนี้จะรอจนกว่าจะมี message เข้ามา

ตัวอย่างการรับ message มีดังนี้

```
#define MAXMSGDATA 4000

msgrcv(id, (char *) &(mesgptr->mesg_type), MAXMESGDATA, mesgptr->mesg_type, 0)
```

เมื่อใช้งาน Message queue เสร็จแล้วเราสามารถลบ Message queue นั้นได้ด้วยคำสั่ง

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl(int msqid, IPC_RMID, struct msqid_ds *buff);
```

- msqid เป็น ตัวบอกว่าเป็น Message queue อันไหน
- IPC_RMID เป็นตัวบอกว่าจะทำการลบ
- struct msqid_ds เป็นชื่อ structure ของ Message queue ในแก่นของระบบ จะต้องกำหนดให้เป็น 0

การลบ Message queue จะทำได้จากทางด้าน server หรือ client ก็ได้ (ใช้เพียงครั้งเดียว)

ตัวอย่างการลบ Message queue มีดังนี้

```
msgctl(id, IPC_RMID, (struct msqid_ds *) 0);
```

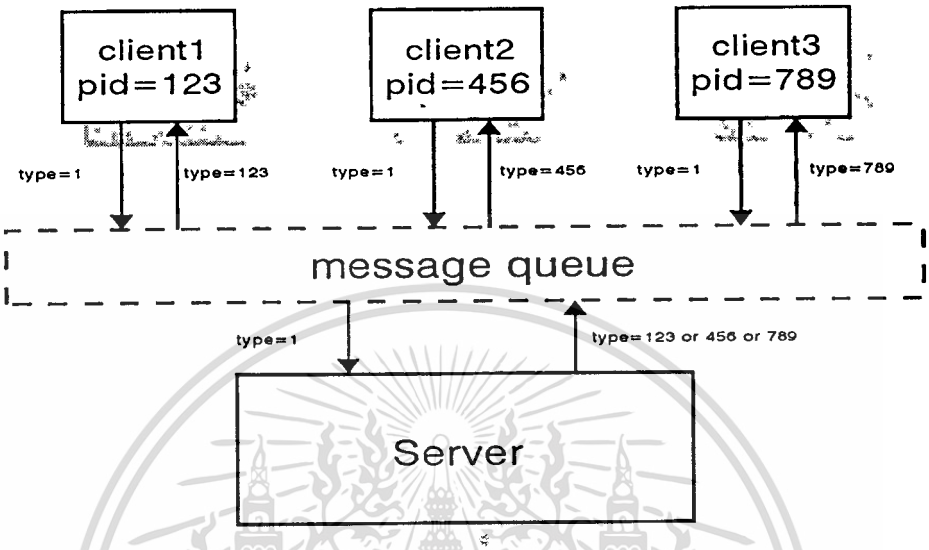
จากคำสั่งที่ผ่านมาเราจะได้ขั้นตอนการใช้งานของ Message queue ดังนี้

เวลา	Server	Client
↓	สร้าง Message queue โดยคำสั่ง msgget (flag = PERMS | IPC_CREAT)	
		ติดต่อกับ Message queue โดยคำสั่ง msgget (flag = 0)
	รับส่งข้อมูลโดยคำสั่ง msgsend, หรือ msgrcv	รับส่งข้อมูลโดยคำสั่ง msgsend, หรือ msgrcv
	ลบ Message queue ด้วยคำสั่ง msgctrl	

4. การติดต่อระหว่าง XServer กับ XClient หลายๆโปรเซส ในการทำงานของ XServer นั้นจะไม่ได้ติดต่อกับ XClient เพียงโปรเซสเดียวเท่านั้น โปรเซสXClient อาจจะมีมากกว่า 1 ก็ได้ ดังนั้นเราจึงต้องมีวิธีที่จะให้ XServer ทราบได้ว่า message ใดเป็นของ XClient ตัวใด

จากที่เราได้ทราบมาแล้วว่า message แต่ละ message จะสามารถระบุ type ได้ซึ่งในการรับเราก็จะสามารถระบุได้เช่นกันว่าจะรับ message ที่มี type ไหน ดังนั้นเราจึงสามารถใช้ type ของ messageนี้สำหรับการแยก message ที่มาจากแต่ละ XClient ได้โดยการระบุให้ message ที่มาจากแต่ละ XClientมี type ที่แตกต่างกัน ซึ่งจะต้องไม่เหมือนกันในแต่ละตัวด้วย

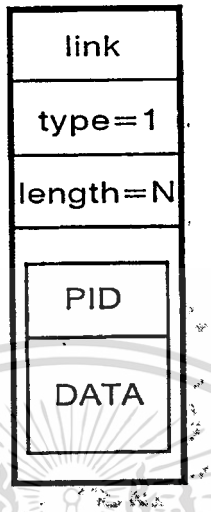
วิธีที่ใช้ในการระบุ type ของ message ในโครงงานนี้คือ ใช้หมายเลขโปรเซส (Process ID) ของแต่ละโปรเซสเป็นค่า type ของ message ดังรูปที่ 33.



รูปที่ 33. แสดงการติดต่อของ XServer กับ XClient หลายๆโปรเซส

จากรูปที่ 33. มี XClient 3 โปรเซส โดยมีหมายเลขโปรเซส 123, 456, 789 ตามลำดับซึ่งมีความหมายว่า XClient แต่ละตัวจะสามารถรับ message ที่มี type เหมือนกับหมายเลขโปรเซสตัวเองได้เท่านั้น เช่น XClient1 จะรับ message ที่มี type เป็น 123 XClient2 จะรับ message ที่มี type เป็น 456 และ XClient3 จะรับ message ที่มี type เป็น 789 เป็นต้น

เรากำหนดให้ XServer รับเฉพาะ message ที่มี type เป็น 1 เสมอ เมื่อ XClient ต้องการที่จะส่งข้อมูลให้กับ XServer ก็จะทำการหาค่าหมายเลขโปรเซสของตนเองแล้วใส่ไว้ในส่วนหัวของข้อมูลที่จะส่ง และกำหนด message type ให้เป็น 1 ดังรูปที่ 34.



รูปที่ 34. แสดงโครงสร้างของ message ที่ XClient ส่งให้ XServer

ซึ่ง message นี้จะอยู่ภายในโครงสร้างของ message ใน Message queue อีกทีหนึ่ง

เมื่อ XServer ได้รับ message จาก XClient แล้วก็จะนำค่าหมายเลขโปรเซสของ XClient ที่เป็นเจ้าของ message นั้นมาเก็บไว้ ซึ่ง message ของ XClient นั้นก็คือการส่งค่าตัวแปรเพื่อให้ XServer ทำการอนุมานนั่นเอง เช่นเมื่อ XServer ได้หาคำตอบจากการอนุมานแล้วก็จะนำค่าหมายเลขโปรเซสนั้นมาเป็น type ของ message และส่งเข้าไปใน Message queue

เราสามารถเขียนแผนภาพแสดงขั้นตอนการทำงานระหว่าง XServer และ XClient ได้ดังนี้

เวลา XServer XClient

1.สร้าง Message queue

2.รอรับข้อมูลจาก Message queue

1.ติดต่อกับ Message queue ที่ XServer สร้างไว้แล้ว

2.หาหมายเลขโปรเซสของตนเอง

3.นำหมายเลขโปรเซสไปใส่ในส่วนหัวของข้อมูล

4.ส่งข้อมูลไปให้ XServer โดยกำหนด type = 1

5.รอรับคำตอบจาก XServer

3.ได้รับข้อมูลที่ type = 1 จาก Message queue

4.เก็บค่าหมายเลขโปรเซสของผู้ส่งจากส่วน
หัวของข้อมูล

5.นำข้อมูลที่ได้ไปทำการอนุมาน

6.ส่งผลอนุมานที่ได้ลงใน Message queue

7.กลับไปทำขั้นตอนที่ 2. ของ XServer ใหม่
จนกว่าจะได้รับคำสั่งให้หยุดการทำงาน

6.ได้รับคำตอบจาก XServer นำคำตอบที่ได้ไปใช้
งานตามที่ต้องการ

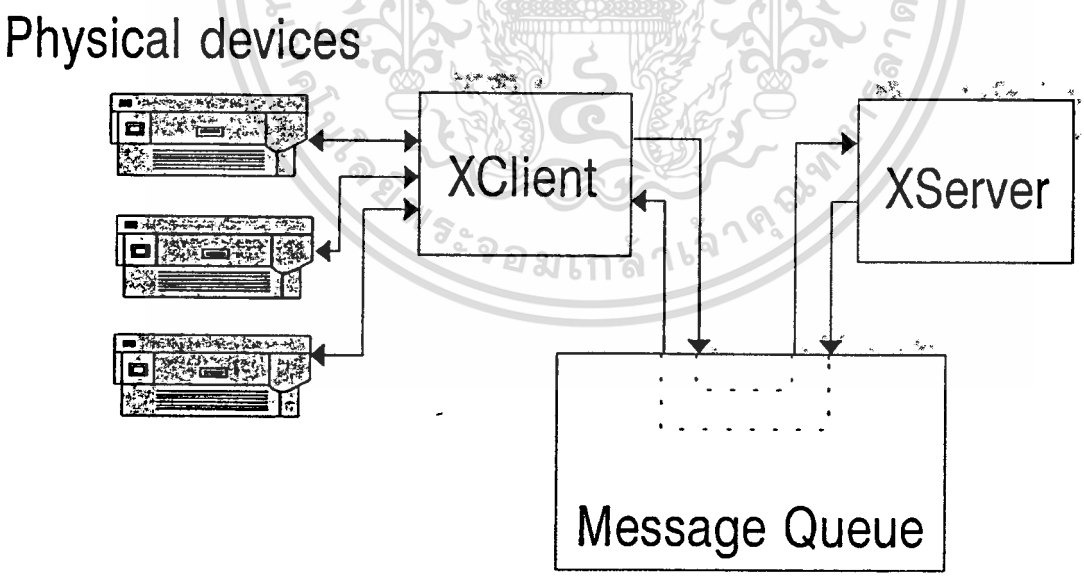
7.เมื่อไม่ต้องการติดต่อกับ XServer อีกให้ ลบ Message queue ถ้าต้องการทำงานต่อ กลับไปที่ขั้นตอนที่ 4. ของ XClient

บทที่ 6

ส่วน XClient

การทำงานของ XClient

XClient เป็นโปรเซสที่คอยขอใช้บริการของ XServer ที่ทำหน้าที่อ่านข้อมูลจากอุปกรณ์ต่างๆ เช่น คีย์บอร์ด เซ็นเซอร์วัดอุณหภูมิ หรือเครื่องชั่งน้ำหนัก เป็นต้น ซึ่ง ค่าที่อ่านได้นี้ XClient จะนำไปแปลงเป็นรูปแบบที่เหมาะสมเพื่อส่งให้กับ XServer ต่อไปดังรูปที่ 35.



ภาพที่ 35. แสดงการทำงานของ XClient

โปรเซส XClient สามารถเป็นได้ทั้งโปรเซสที่ทำงานอยู่เบื้องหน้า (foreground process) หรือโปรเซสที่ทำงานอยู่เบื้องหลัง (background process) ก็ได้ ขั้นตอนการทำงานของ XClient จะมีดังนี้คือ

- รับข้อมูลจากอุปกรณ์อินพุท
- แปลงเป็นรูปแบบที่เหมาะสม
- ส่งข้อมูลที่แปลงรูปแบบแล้วลงใน Message queue
- รอรับการส่งคำตอบกลับจาก XServer
- กลับไปรับข้อมูลจากอุปกรณ์อินพุทใหม่จนกว่าจะ exit

การรับข้อมูลจากอุปกรณ์อินพุทนี้เป็นการอ่านค่าโดยใช้คำสั่งที่เหมาะสมกับอุปกรณ์นั้นๆ เช่น ถ้าอุปกรณ์อินพุทเป็น คีย์บอร์ด เราสามารถอ่านค่าได้โดยใช้คำสั่ง fgets, fread, ... ในภาษาซี ถ้าอุปกรณ์อินพุทมีการต่อเข้ามาที่เครื่องที่ XServer ทำงานอยู่โดยผ่านทาง TCP/IP network ก็อ่านค่าเข้ามาได้ด้วยคำสั่ง read เป็นต้น

เมื่อมีการอ่านค่าเข้ามาแล้วหน้าที่ต่อไปของ XClient คือการแปลงรูปแบบ (format) ของข้อมูลที่ได้รับมานั้นเป็นรูปแบบที่ XServer รู้จัก เหตุที่ต้องมีการแปลงรูปแบบนั้นเป็นเพราะ XServer จะรับค่าอินพุทเป็น string ข้อความว่าตัวแปรใดมีค่าเป็นเท่าไร เช่น "TEMP=100" แต่อุปกรณ์ต่างๆที่นำมาต่อเข้ากับระบบเช่น sensor ต่างๆนั้นส่วนใหญ่จะมีความซับซ้อนน้อยและจะส่งค่าได้เป็นตัวเลขว่าค่าที่อุปกรณ์นั้นตรวจจับได้เป็นเท่าไร เช่น sensor วัดอุณหภูมิอาจจะส่งค่าเป็นตัวเลข 8 บิททุกๆ 10 วินาทีเพื่อรายงานค่าอุณหภูมิขณะนั้น โดย XClient จะทำหน้าที่ดูว่าข้อมูลที่เข้ามานั้นมาจากอุปกรณ์อะไรเพื่อที่จะได้ทราบว่าข้อมูลที่เข้ามานั้นเป็นข้อมูลของตัวแปรอะไร โปรเซส XClient นี้จะติดต่อกับอุปกรณ์เพียงอย่างเดียวหรือหลายอย่างก็ได้ แล้วแต่ความต้องการของผู้ใช้ เพียงแต่ในโปรแกรมต้องทราบว่าข้อมูลที่เข้ามาเป็นของอุปกรณ์ใดและอุปกรณ์นั้นๆเป็นอุปกรณ์ที่เกี่ยวข้องกับตัวแปรใด

เมื่อ XClient แปลงข้อมูลให้อยู่ในรูปที่ XServer รู้จักแล้วก็จะนำข้อมูลนั้นใส่ลงใน Message queue เพื่อที่จะให้ XServer มารับไปอนุมานต่อไป รายละเอียดของการส่งและรับข้อมูลกับ Message queue นั้นจะกล่าวถึงในหัวข้อถัดไป

หลังจากที่ XServer ได้ทำการอนุมานเสร็จแล้วจะส่งคำตอบซึ่งเป็น string ข้อความว่าได้ผลสรุปของการอนุมานอย่างไรมาให้กับ XClient โดยที่ ผู้เขียนโปรแกรม XClient สามารถจะนำค่าของคำตอบนี้ไปทำการควบคุมอุปกรณ์อื่นๆได้ว่าจะต้องการให้ทำอะไรกับคำตอบที่ได้มาตัวอย่าง

เช่นถ้าสมมติว่าเรามีระบบเครื่องยนต์ที่ต้องใช้น้ำเป็นตัวระบายความร้อนและมีอุปกรณ์ตรวจจับอุณหภูมิว่าถ้าอุณหภูมิของน้ำที่ใช้ระบายความร้อนให้เครื่องยนต์มีค่ามากกว่า 100 องศาให้ทำการเปิดวาล์วน้ำให้ระบายนำร้อนออก และนำน้ำเย็นเข้ามา เราจะเขียนกฎง่ายๆได้ดังนี้

if (TEMP > 100) then (HOTVALVE=open)&&(COOLVALVE=open)

โดยที่

TEMP เป็นค่าตัวแปรที่บอกอุณหภูมิของน้ำในขณะนั้น

VALVE เป็นตัวบอกสถานะของ valve น้ำว่าปิดหรือเปิด

และถ้า XClient ส่งค่า "TEMP=100" ให้กับ XServer แล้ว เมื่อ XServer ทำการอนุมานเสร็จจะได้คำตอบว่า "VALVE=open" จากนั้น XServer ก็จะส่งค่า "VALVE=open" กลับมาให้กับ XClient โปรแกรม XClient ก็จะสามารถรู้ได้ว่าจะต้องทำการเปิด valve ให้นำน้ำร้อนออกและเปิด valve ให้น้ำเย็นเข้า เป็นต้น

ส่วนเชื่อมต่อระหว่าง XClient กับ XServer

XCient จะติดต่อกับ XServer โดยผ่านทาง IPC โดยก่อนที่ XClient จะใช้งาน Message queue ได้นั้นทาง XServer จะต้องสร้าง Message queue นั้นๆขึ้นมาก่อน แล้วทาง XClient จึงจะสามารถติดต่อกับ Message queue นั้นๆได้ สำหรับคำสั่งที่ XClient ใช้สำหรับการติดต่อกับ Message queue ก็คือคำสั่ง "msgget" ดังนี้

id=msgget(MKEY1, 0);

โดยที่ MKEY คือชื่อของ Message queue

คำสั่งนี้จะทำการติดต่อกับ Message queue เท่านั้น ไม่ได้ทำการสร้าง Message queue ขึ้นมา โดยที่คำสั่ง "msgget" จะส่งค่ากลับมาเป็นเลขจำนวนเต็ม (id) เพื่อใช้เป็นเลขแทน Message queue นั้นๆ ตัวอย่างเช่นในการส่งข้อมูลลงใน Message queue ทำได้โดย คำสั่ง "msg_send" ดังนี้

```
msg_send(id, &mesg);
```

หมายถึงให้ส่งข้อมูลที่อยู่ในตัวแปร mesg ให้กับ Message queue ที่มีหมายเลขประจำเป็นค่าของข้อมูลในตัวแปรชื่อ id

สำหรับรายละเอียดของคำสั่งต่างๆที่ใช้ได้สำหรับ XClient จะเหมือนกับ คำสั่งที่ใช้กับ XServer

ตัวอย่างโปรแกรมสำหรับเชื่อมต่อกับเปลือกระบบผู้เชี่ยวชาญ

โปรแกรมต่อไปนี้เป็นตัวอย่างโปรแกรมของ XClient ที่เขียนโดยใช้ sub-routine และ include file ที่ระบบให้มา

```
#include <stdio.h>
#include "Xlib.h"

Mesg mesg;
main()
{  int  id;

        xopenq(&id);          /* สร้าง ทางติดต่อกับ Message queue ของ XServer  */
        client(id);           /*  โปรแกรมอ่านค่าจากอุปกรณ์รับข้อมูล          */
        mesg_send(id, &mesg);       /*     ส่งข้อมูลลงใน Message queue ให้กับ
    XServer  */
        xcloseq(&id);         /*  ยกเลิกทางติดต่อกับ Message queue ของ XServer */
        exit(0);
}
```

```
client(id)
int id;
{       int  n;
                if(fgets (mesg.mesg_data, MAXMESGDATA, stdin) == NULL)        /*
อ่านค่าจาก  Standard Input  */
                printf("read error\n");
                n = strlen(mesg.mesg_data);
                if(mesg.mesg_data[n-1] == '\n')
                n--;
                mesg.mesg_data[n] = '\0';
                mesg.mesg_len = n;
                mesg.mesg_type = 1L;
}
```

บทที่ 7

ตัวอย่างการใช้งาน

การใช้งานโปรแกรม XAdmin

หมายเหตุ : ตัวอักษร **ตัวเข้ม** คือคำสั่งที่ผู้ใช้ต้องพิมพ์เข้าไป

ตัวอักษร *ตัวเอียง* คือข้อมูลที่โปรแกรมแสดงผลออกมา

- ไฟล์ที่ต้องการ

 # ls -l

-rwxr-xr-x 1 root other 86468 Jul 5 16:47 xadmin

-rwxr-xr-x 1 root other 77960 Jul 5 15:54 xsrv

- การเรียกใช้งาน XAdmin

โปรแกรม XAdmin ซึ่งทำหน้าที่ในการควบคุมการทำงานของระบบทั้งหมดสามารถเรียกใช้งานได้

โดยพิมพ์คำสั่ง "xadmin" ที่ prompt ของยูนิกซ์ คำสั่งต่อไปนี้ แสดงการเรียกใช้งานโปรแกรม XAdmin

และผลลัพธ์ที่ได้จากการเรียกโปรแกรมนี้

 # xadmin

Welcome to I/O Independent Expert System Shell

Initialize...

Processing [A.cvl]

Processing [B.cvl]

Processing [C.cvl]

Processing [D.cvl]

Processing [E.cvl]

Processing [F.cvl]

Processing [G.cvl]

Processing [H.cvl]

Processing [I.cvl]

Processing [J.cvl]

Processing [K.cvl]

Processing [L.cvl]

Processing [M.cvl]

Processing [N.cvl]

Processing [O.cvl]

Processing [P.cvl]

Processing [Q.cvl]

Processing [R.cvl]

Processing [S.cvl]

Processing [T.cvl]

Processing [U.cvl]

Processing [V.cvl]

Processing [W.cvl]

Processing [X.cvl]

Processing [Y.cvl]

Processing [Z.cvl]

Initialize complete

[xAdmin] :

เมื่อเรียกโปรแกรม XAdmin ขึ้นมาแล้วจะปรากฏ prompt ของโปรแกรม XAdmin เพื่อรอรับคำสั่งจากผู้ใช้

- การขอความช่วยเหลือ (Help) ในการเรียกใช้คำสั่งต่างๆของโปรแกรม XAdmin

เมื่อพิมพ์คำว่า "help" ที่ prompt ของ XAdmin จะเป็นการพิมพ์คำสั่งทั้งหมดของโปรแกรมพร้อมคำอธิบายขึ้นบนหน้าจอ

[xAdmin] : **help**

Help Subsystem

delrule	*: Mark a rule deleted*
displayfact	*: Print value of all fact*
help	*: Provide this screen*
loadfact	*: Load default facts from file*
newfact	*: Enter a new fact*
newrule	*: Enter new rule*
udelrule	*: Undelete rule*
packrule	*: Delete all deleted mark rule*
rulelist	*: List all rules*
savefact	*: Save default facts to file*
startsrv	*: Start Expert System Server*
stopsrv	*: Stop Expert System Server*
why	*: Explan how system produce the advice*
xping	*: Ping to XServer*
exit	*: Quit from this program*

คำสั่งของโปรแกรมจะอยู่ทางด้านขวามือ ส่วนทางด้านซ้ายมือจะเป็นคำอธิบายของคำสั่งนั้นๆ

- การเพิ่มกฎใหม่ให้กับระบบ

การเพิ่มกฎใหม่ให้กับระบบนี้ก็คือการใส่ความรู้เพิ่มให้กับโปรแกรม กฎที่ใส่ให้กับระบบมีรูปแบบดังนี้คือ

if VARIABLE1 = VALUE1 then VARIABLE2 = VALUE2

-VARIABLE1, VARIABLE2 คือชื่อตัวแปรซึ่งต้องเป็นอักษรตัวใหญ่หมดและมีความยาวไม่เกิน 8 ตัวอักษร

ตัวอย่างของชื่อตัวแปรที่ถูกต้องคือ TEMP, QTY, LENGTH, RESULT

ตัวอย่างของชื่อตัวแปรที่ผิดคือ

Quality : เพราะมีอักษรตัวเล็กอยู่ในตัวแปร

TEMPERATURE : เพราะความยาวเกิน 8 ตัวอักษร

RESULT_1 : เพราะมีอักขระที่ไม่ใช่ตัวอักษรตัวใหญ่อยู่ในชื่อตัวแปร

-VALUE1, VALUE2 คือค่าของตัวแปรชนิด string

ตัวอย่างของกฎที่ถูกต้องมีดังนี้

if TEMP=1 then SWITCHA=off

if AAA=2 then BBB=3

if RESULT=yes then REPLY=50

ภายในกฎสามารถมีเครื่องหมายทางตรรกศาสตร์ logic ได้คือ เครื่องหมาย "หรือ (or)" และ เครื่องหมาย "และ (and)" ดังนี้

เครื่องหมาย	สัญลักษณ์
หรือ	|
และ	&

[xAdmin] : **newrule**

Please enter rule

: if(INTEREST=fall) then (STOCKS=rise)

Please enter rule

: **if(INTEREST=rise) then (STOCKS=fall)**

Please enter rule

: **if(DOLLAR=fall) then (INTEREST=rise)**

Please enter rule

: **if(DOLLAR=rise) then (INTEREST=fall)**

Please enter rule

: **if(FEDINT=fall)&&(FEDMON=add) then (INTEREST=fall)**

Please enter rule

: **exit**

- การขอดูกฎที่ระบบมีอยู่

เราสามารถเรียกดูกฎที่มีอยู่ทั้งหมดในฐานความรู้ได้โดยใช้คำสั่ง rulelist ซึ่งจะได้ผลลัพธ์ดังนี้

[xAdmin] : **rulelist**

RULE LIST

1).[U] if (INTEREST=fall) then (STOCKS=rise)

2).[U] if (INTEREST=rise) then (STOCKS=fall)

3).[U] if(DOLLAR=fall) then (INTEREST=rise)

4).[U] if(DOLLAR=rise) then (INTEREST=fall)

5).[U] if(FEDINT=fall)&&(FEDMON=add) then (INTEREST=fall)

หมายเลขข้างหน้าคือหมายเลขที่ของกฎนั้นๆซึ่งจะใช้เป็นตัวอ้างอิงในการลบกฎ หรือการเรียกคืน ซึ่งจะได้กล่าวในหัวข้อต่อไป ตัวอักษรที่อยู่ในเครื่องหมาย [ ] คือ flag ของกฎนั้นๆโดยมีความหมายดังนี้คือ

U หมายถึงเครื่องอนุมาน (Inference Engine) สามารถนำกฎนี้สามารถใช้งานได้

D หมายถึงเครื่องอนุมานจะไม่นำกฎนี้ไปใช้งาน

- การลบกฎที่ไม่ต้องการออกจากระบบ

เมื่อเราต้องการให้เครื่องอนุมานไม่พิจารณากฎใด เราสามารถทำได้โดยใช้คำสั่ง delrule ซึ่งการใช้งานทำได้ดังนี้

[xAdmin] : **rulelist**

RULE LIST

1).[U] if (INTEREST=fall) then (STOCKS=rise)

2).[U] if (INTEREST=rise) then (STOCKS=fall)

3).[U] if(DOLLAR=fall) then (INTEREST=rise)

4).[U] if(DOLLAR=rise) then (INTEREST=fall)

5).[U] if(FEDINT=fall)&&(FEDMON=add) then (INTEREST=fall)

[xAdmin] : **delrule**

DELETE RULE

Please enter number of rule : **3**

[xAdmin] : **rulelist**

RULE LIST

1).[U] if (INTEREST=fall) then (STOCKS=rise)

2).[U] if (INTEREST=rise) then (STOCKS=fall)

3).[D] if(DOLLAR=fall)□then (INTEREST=rise)

4).[U] if(DOLLAR=rise) then (INTEREST=fall)

5).[U] if(FEDINT=fall)&&(FEDMON=add) then (INTEREST=fall)

การลบแบบนี้เป็นเพียงการกำหนดค่า flag ของกฎนั้นๆให้เป็นแบบ Delete เพื่อไม่ให้เครื่องอนุมาน นำกฎนี้ไปใช้ แต่ตัวกฎยังคงอยู่ในฐานความรู้ ถ้าต้องการนำกฎนี้กลับมาใช้ใหม่ทำได้โดยใช้คำสั่ง undelrule ดังนี้

[xAdmin] : **rulelist**

RULE LIST

1).[U] if (INTEREST=fall) then (STOCKS=rise)

2).[U] if (INTEREST=rise) then (STOCKS=fall)

3).[D] if(DOLLAR=fall)□then (INTEREST=rise)

4).[U] if(DOLLAR=rise) then (INTEREST=fall)

5).[U] if(FEDINT=fall)&&(FEDMON=add) then (INTEREST=fall)

[xAdmin] : **udelrule**

UNDELETE RULE

Please enter number of rule : **3**

[xAdmin] : rulelist

RULE LIST

1).[U] if (INTEREST=fall) then (STOCKS=rise)

2).[U] if (INTEREST=rise) then (STOCKS=fall)

3).[U] if(DOLLAR=fall) then (INTEREST=rise)

4).[U] if(DOLLAR=rise) then (INTEREST=fall)

5).[U] if(FEDINT=fall)&&(FEDMON=add) then (INTEREST=fall)

- การลบกฎที่ไม่ต้องการออกจากระบบอย่างถาวร

เป็นการลบกฎที่ได้ทำการลบโดยใช้คำสั่ง delrule แล้วออกจากฐานความรู้ อย่างถาวร เนื่องจาก คำสั่ง delrule จะไม่ลบกฎนั้นออกจากฐานความรู้จึงทำให้สิ้นเปลืองเนื้อที่ในการเก็บ คำสั่งนี้จะทำการลบกฎที่มี flag เป็น "D" (Delete) ออกจากแฟ้มข้อมูล และเลื่อนกฎถัดไปขึ้นมาแทนที่ รวมทั้งจัดทำ Index สำหรับแฟ้มข้อมูลนั้นใหม่ สำหรับการทำงานดังที่ได้กล่าวมานี้ทำได้โดยใช้คำสั่ง packrule ซึ่งจะได้ผลลัพธ์ดังนี้

[xAdmin] : **rulelist**

RULE LIST

1).[U] if (INTEREST=fall) then (STOCKS=rise)

2).[U] if (INTEREST=rise) then (STOCKS=fall)

3).[U] if(DOLLAR=fall)□then (INTEREST=rise)

4).[U] if(DOLLAR=rise) then (INTEREST=fall)

5).[U] if(FEDINT=fall)&&(FEDMON=add) then (INTEREST=fall)

[xAdmin] **: delrule**

DELETE RULE

Please enter number of rule : **3**

[xAdmin] : **packrule**

PACK RULE

Done

[xAdmin] : **rulelist**

RULE LIST

1).[U] if (INTEREST=fall) then (STOCKS=rise)

2).[U] if (INTEREST=rise) then (STOCKS=fall)

3).[U] if(DOLLAR=rise) then (INTEREST=fall)

4).[U] if(FEDINT=fall)&&(FEDMON=add) then (INTEREST=fall)

- การสั่งให้ eXpert system Server (XServer) เริ่มทำงาน

เราสามารถสั่งให้ XServer ทำงานได้โดยใช้คำสั่ง startsrv ซึ่งจะได้ผลลัพธ์ดังนี้

[xAdmin] : **startsrv**

START EXPERT SYSTEM SERVER

Sending output to nohup.out

[xAdmin] : **exit**

 # ps -e

PID	*TTY*	*TIME*	*COMD*
0	*?*	*0:03*	*sched*
1	*?*	*8:19*	*init*
2	*?*	*0:00*	*pageout*
3	*?*	*66:30*	*fsflush*
202	*?*	*0:10*	*sac*
203	*console*	*0:01*	*ttymon*
176	*?*	*0:00*	*lpNet*
106	*?*	*0:04*	*rpcbind*

```
123 ?       0:03    inetd
98 ?        1:51    in.route
108 ?       0:00    keyserv
130 ?       0:00    automoun
114 ?       0:00    kerbd
134 ?       0:01    statd
136 ?       0:02    lockd
147 ?       0:01    syslogd
167 ?       0:00    lpsched
157 ?       0:25    cron
175 ?       0:06    sendmail
206 ?       0:11    ttymon
200 ?       0:02    vold
184 ?       0:35    utmpd
6143 pts/0  0:01    script
6136 pts/0  0:01    sh
6167 pts/2  0:00    ps
6142 pts/0  0:01    script
6129 ?      0:0     4 in.telne
6152 pts/2  0:00    sh
6163 pts/2  0:00    xsrv    <---------- โปรเซสของ eXpert system Server
```

หลังจากคำสั่งนี้แล้วโปรแกรม XAdmin จะสร้างโปรเซสขึ้นมาใหม่ชื่อว่า xsrv สำหรับประมวลผล

- การตรวจสอบการทำงานของ XServer

เมื่อเราต้องการทราบว่า XServer ทำงานหรือไม่นั้นเราสามารถตรวจสอบได้โดยการใส่ข้อความ (message) ไปยัง XServer ซึ่งเมื่อ XServer ได้รับแล้วก็จะส่งข้อความกลับมาให้กับ XAdmin ถ้าเราได้รับ reply message จาก XServer หมายความว่า XServer ทำงานอยู่ แต่ถ้าเกิด

timeout แสดงว่า XServer ยังไม่ได้เริ่มทำงาน การตรวจสอบนี้ทำได้โดยใช้คำสั่ง xping จะได้ผลลัพธ์ดังนี้

[xAdmin] : **startsrv**

START EXPERT SYSTEM SERVER

Sending output to nohup.out

[xAdmin] : **exit**

 # ps -e

PID TTY TIME COMD

0 ? 0:03 sched

1 ? 8:19 init

2 ? 0:00 pageout

3 ? 66:30 fsflush

.....

6231 ? 0:00 xsrv <---------- โปรเซสของ eXpert system Server

 # xadmin

Welcome to I/O Independent Expert System Shell

Initialize...

Processing [A.cvl]

Processing [B.cvl]

.....

Processing [Y.cvl]

Processing [Z.cvl]

Initialize complete

[xAdmin] : **xping**

XPING

XServ is alive

[xAdmin] : **exit**

#ps -e | grep srv

6231 ? 0:00 xsrv

 # kill 6163

 # ps -e | grep xsrv

 # xadmin

Welcome to I/O Independent Expert System Shell

Initialize...

Processing [A.cvl]

Processing [B.cvl]

.....

Processing [Y.cvl]

Processing [Z.cvl]

Initialize complete

[xAdmin] : **xping**

XPING

cannot get Message queue

- การใส่ข้อมูลให้กับ XServer เพื่อให้ XServer ประมวลผล

ในโปรแกรม XAdmin นั้นมีโปรแกรม XClient ซึ่งรับข้อมูลจากทางแป้นพิมพ์อยู่ด้วย ทางผู้ใช้จึง สามารถใส่ข้อมูลทางโปรแกรม XAdmin เพื่อให้ XServer นำไปใช้ประมวลผลได้ โดยผลลัพธ์ที่ ได้จะถูกแสดงออกทางจอภาพ

เราสามารถใส่ข้อมูลให้กับ XServer ได้โดยใช้คำสั่ง newfact ดังตัวอย่างต่อไปนี้

RULE LIST

1).[U] if (INTEREST=fall) then (STOCKS=rise)

2).[U] if (INTEREST=rise) then (STOCKS=fall)

3).[U] if(DOLLAR=fall) then (INTEREST=rise)

4).[U] if(DOLLAR=rise) then (INTEREST=fall)

5).[U] if(FEDINT=fall)&&(FEDMON=add) then (INTEREST=fall)

[xAdmin] : **newfact**

INPUT A FACT

Please enter fact: **DOLLAR=rise**

OUTPUT

INTEREST=fall

- การขอคำอธิบายว่าคำตอบที่ XServer สรุปให้นั้นได้มาอย่างไร

ในบางครั้งผู้ใช้ต้องการทราบว่าคำตอบที่ได้จาก XServer นั้นได้มาอย่างไร มีกฎข้อใด บ้างที่ถูกนำมาใช้ ผู้ใช้สามารถเรียกดูได้จากคำสั่ง why ดังตัวอย่างต่อไปนี้

RULE LIST

1).[U] if (INTEREST=fall) then (STOCKS=rise)

2).[U] if (INTEREST=rise) then (STOCKS=fall)

3).[U] if(DOLLAR=fall) then (INTEREST=rise)

4).[U] if(DOLLAR=rise) then (INTEREST=fall)

5).[U] if(FEDINT=fall)&&(FEDMON=add) then (INTEREST=fall)

[xAdmin] : **newfact**

INPUT A FACT

Please enter fact: **DOLLAR=rise**

OUTPUT

INTEREST=fall

STOCKS=rise

[xAdmin] : **why**

EXPLANATION

 # Rule : if (DOLLAR = rise) then (INTEREST = fall)

Result : INTEREST=fall

 # Rule : if (INTEREST = fall) then (STOCKS=rise)

Result : STOCKS=rise

- การขอดูค่าของ fact ทั้งหมดที่ XServer ใช้ในการประมวลผล

[xAdmin] : **displayfact**

DISPLAY FACT

DOLLAR=rise

FEDINT=null

FEDMON=null

INTEREST=fall

STOCKS=rise

[xAdmin] : **newfact**

INPUT A FACT

Please enter fact: **DOLLAR=fall**

OUTPUT

INTEREST=rise

STOCKS=fall

[xAdmin] : **displayfact**

DISPLAY FACT

DOLLAR=fall

FEDINT=null

FEDMON=null

INTEREST=rise

STOCKS=fall

- การเก็บค่า fact ของ XServer ลงในแฟ้มข้อมูล

เราสามารถเก็บค่าของ fact ในขณะใดขณะหนึ่งไว้ในแฟ้มข้อมูลเพื่อเก็บไว้ใช้งานภายหลังได้โดยใช้คำสั่ง "savefact"

[xAdmin] : **savefact**

SAVEFACT

Saved

- การอ่านค่าของ fact จากแฟ้มข้อมูลเพื่อตั้งค่าเริ่มต้นให้กับ XServer

คือการอ่านค่าของ fact ที่ได้เก็บไว้ด้วยคำสั่ง savefact มาใช้งานเป็นค่าเริ่มต้นของตัวแปรใน XServer

[xAdmin] : **loadfact**

LOAD FACT

Loaded

- การหยุดการทำงานของ XServer

[xAdmin] : **stopsrv**

STOP SERVER

Stoppped

- การออกจากระบบ XAdmin

[xAdmin] : **exit**

Server Terminated.

บทที่ 8
สรุปและข้อเสนอแนะ

ผลที่ได้จากการพัฒนาโครงงานนี้คือเปลือกระบบผู้เชี่ยวชาญที่ไม่ขึ้นกับอุปกรณ์รับส่งข้อมูลซึ่งเป็นเปลือกระบบผู้เชี่ยวชาญที่แตกต่างจากเดิม กล่าวคือผู้ใช้สามารถปรับแต่งให้เข้ากับการใช้งานกับระบบของตนเองได้ง่ายขึ้นโดยการเขียนโปรแกรมในส่วนที่ใช้ในการเชื่อมต่อกับอุปกรณ์รับส่งข้อมูลที่ระบบนั้นๆต้องการ ซึ่งถ้าผู้ใช้ไม่ต้องการใช้งานอุปกรณ์อื่นๆก็สามารถติดต่อกับระบบผู้เชี่ยวชาญทางแป้นพิมพ์และจอภาพได้เหมือนกับเปลือกระบบผู้เชี่ยวชาญปกติ สำหรับการสร้างโปรแกรมเพื่อเชื่อมต่อกับเปลือกระบบผู้เชี่ยวชาญนั้น จะมีอุปกรณ์อำนวยความสะดวกให้กับผู้ใช้คือจะมีโปรแกรมย่อยต่างๆในการรับส่งข้อมูลที่ผู้ใช้สามารถเรียกใช้ได้โดยสะดวก ทำให้การเขียนโปรแกรมเพื่อเชื่อมต่อกับระบบง่ายขึ้น

ในการทำงานโครงงานนี้พบว่าส่วนที่ยากและมีปัญหามากที่สุดจะมี 3 ส่วนคือ ส่วนควบคุมการติดต่อระหว่างโปรเซส ส่วนแปลความหมาย และส่วนอนุมาน เนื่องจากข้อจำกัดด้านเวลา ดังนั้นในวิทยานิพนธ์นี้จึงใช้วิธีที่ง่ายที่สุดในการแก้ปัญหาของแต่ละเรื่อง

ข้อเสนอแนะที่น่าสนใจคือ ในการพัฒนาต่อไป เราสามารถนำแนวความคิดนี้มาสร้างเปลือกระบบผู้เชี่ยวชาญที่มีส่วนประกอบของ ระบบผู้เชี่ยวชาญ เช่นส่วนอนุมาน ส่วนจัดการฐานความรู้ ส่วนอธิบาย

แยกจากกันเป็นแต่ละโปรเซส ซึ่งจะทำให้เราสามารถเลือกได้ว่า เปลือกระบบผู้เชี่ยวชาญที่เราต้องการใช้นั้นจะมีการอนุมานแบบไหนมีการจัดการฐานความรู้อย่างไร การทำงานของเปลือกระบบผู้เชี่ยวชาญแบบนี้นั้นนอกจากจะไม่ขึ้นกับอุปกรณ์อินพุทเอาท์พุทแล้วเรายังสามารถเลือกส่วนประกอบต่างๆของระบบได้ตามที่เราต้องการด้วย

บรรณานุกรม

ครรชิต ไมตรี. ระบบผู้เชี่ยวชาญ. กรุงเทพฯ:คณะวิศวกรรมศาตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง, 2535.

วิลาศ วูวงศ์ และ บุญเจริญ ศิริเนาวกุล. ระบบผู้เชี่ยวชาญ. กรุงเทพฯ:ศูนย์เทคโนโลยีอิเล็คทรอนิกส์และคอมพิวเตอร์แห่งชาติ สำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ กระทรวงวิทยาศสาสตร์และสิ่งแวดล้อม, 2535.

Ignizio, James P. Introduction to Expert Systems. McGRAW-HILL, 1991.

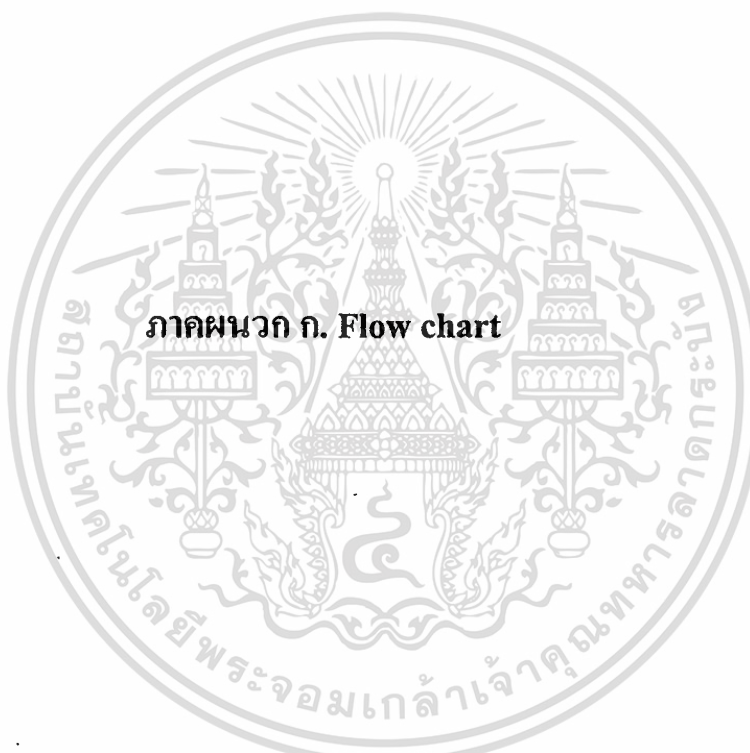
Leigh, William E. and Michael E. Doherty. Decision Support and Expert System. SOUTH-WESTERN PUBLISHING CO., 1986.

Levine, Robert I., Diane E. Drang and Barry Edlson. A Comprehensive Guide to AI and ExpertSystemsUsingTurboPascal. McGRAW-HILL, 1988.

Martin, James and Steven Oxman. Building Expert System. Prentice-Hall, 1988.

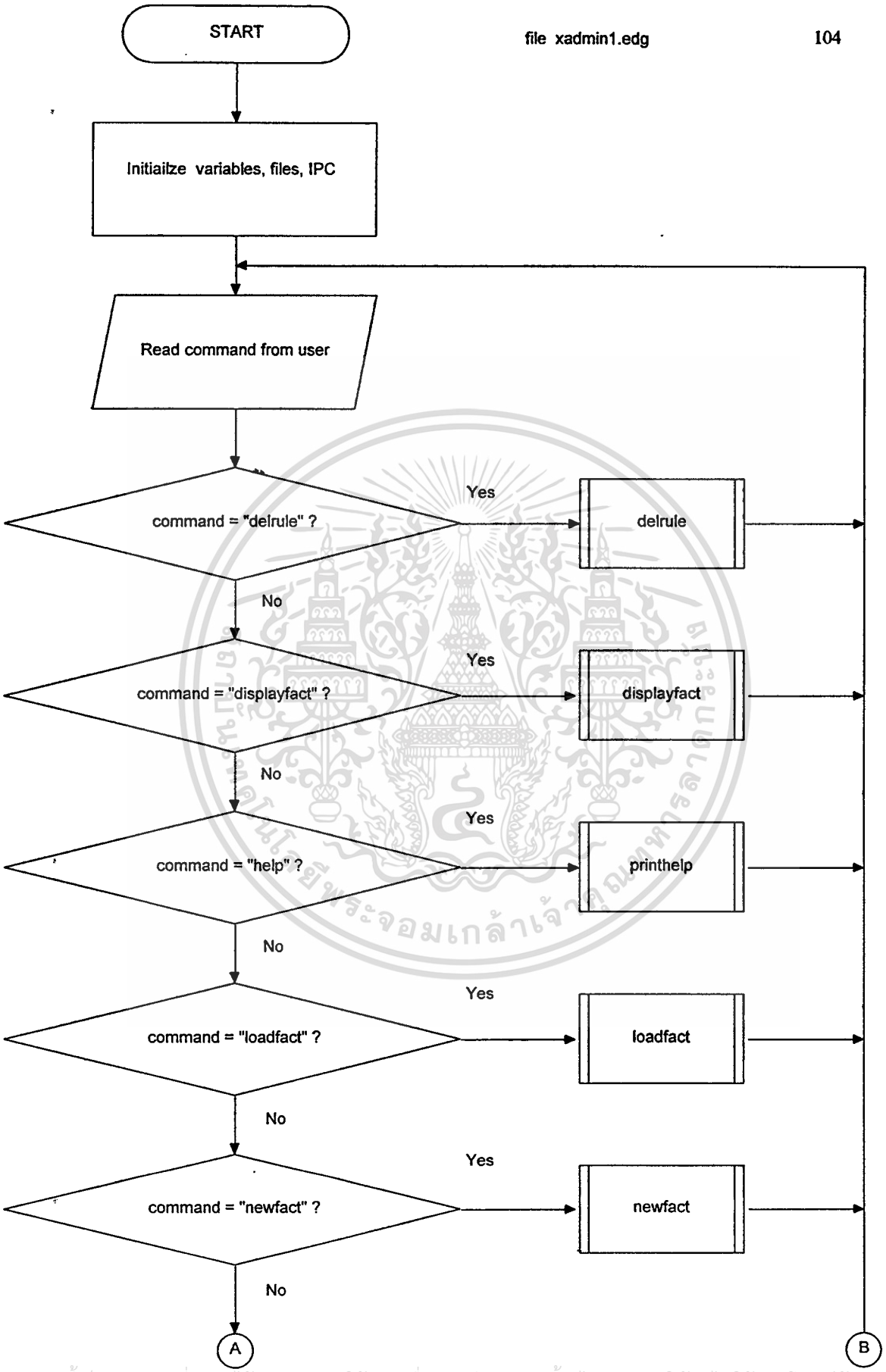
Stevens Richard W. UNIX Network Programming. Prentice-Hall, 1991.

Tremblay, Jean-Paul and Paul G.Sorenson. Compiler Writing. McGRAW-HILL,1985.



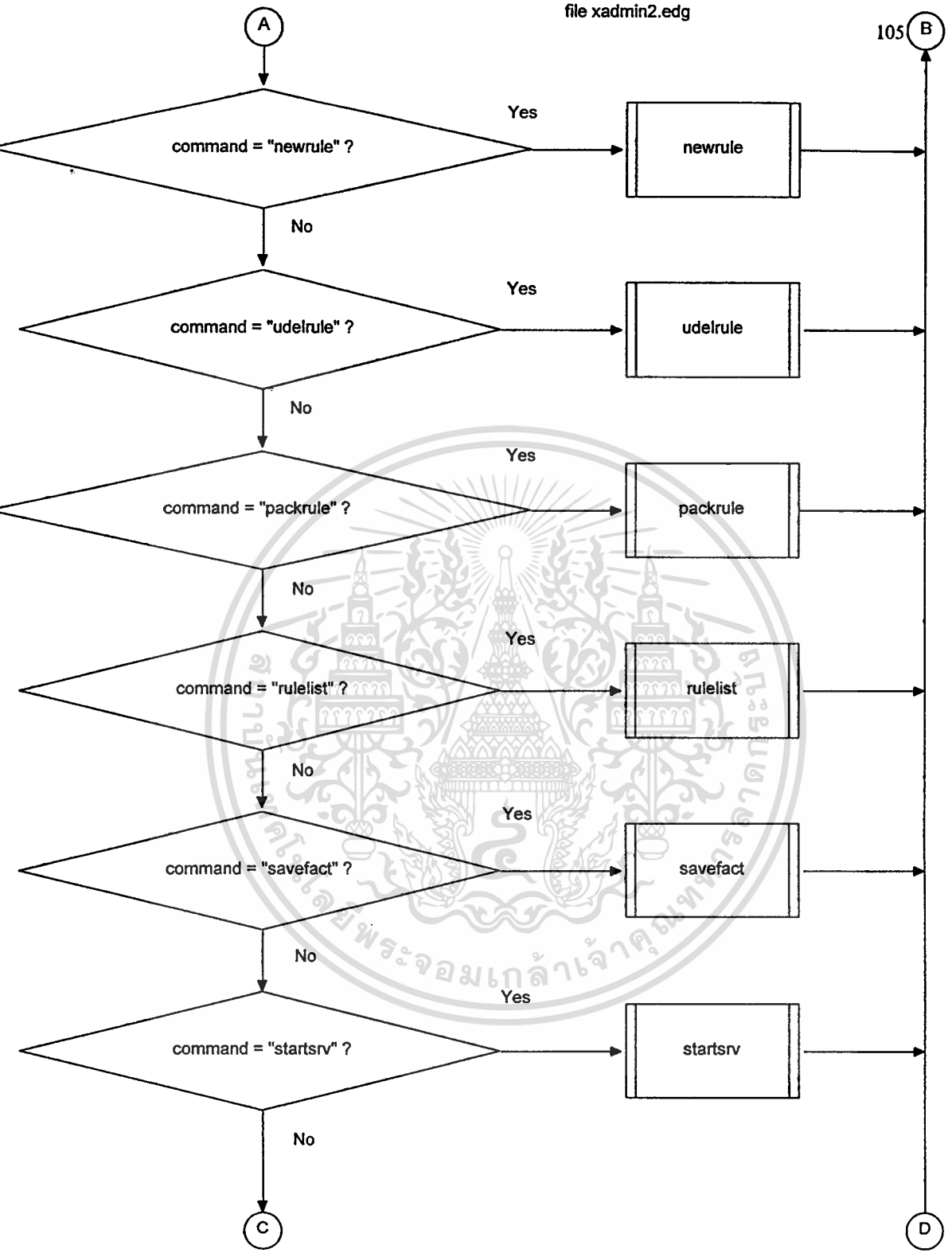
ภาคผนวก ก. Flow chart

XADMIN : main 1/3

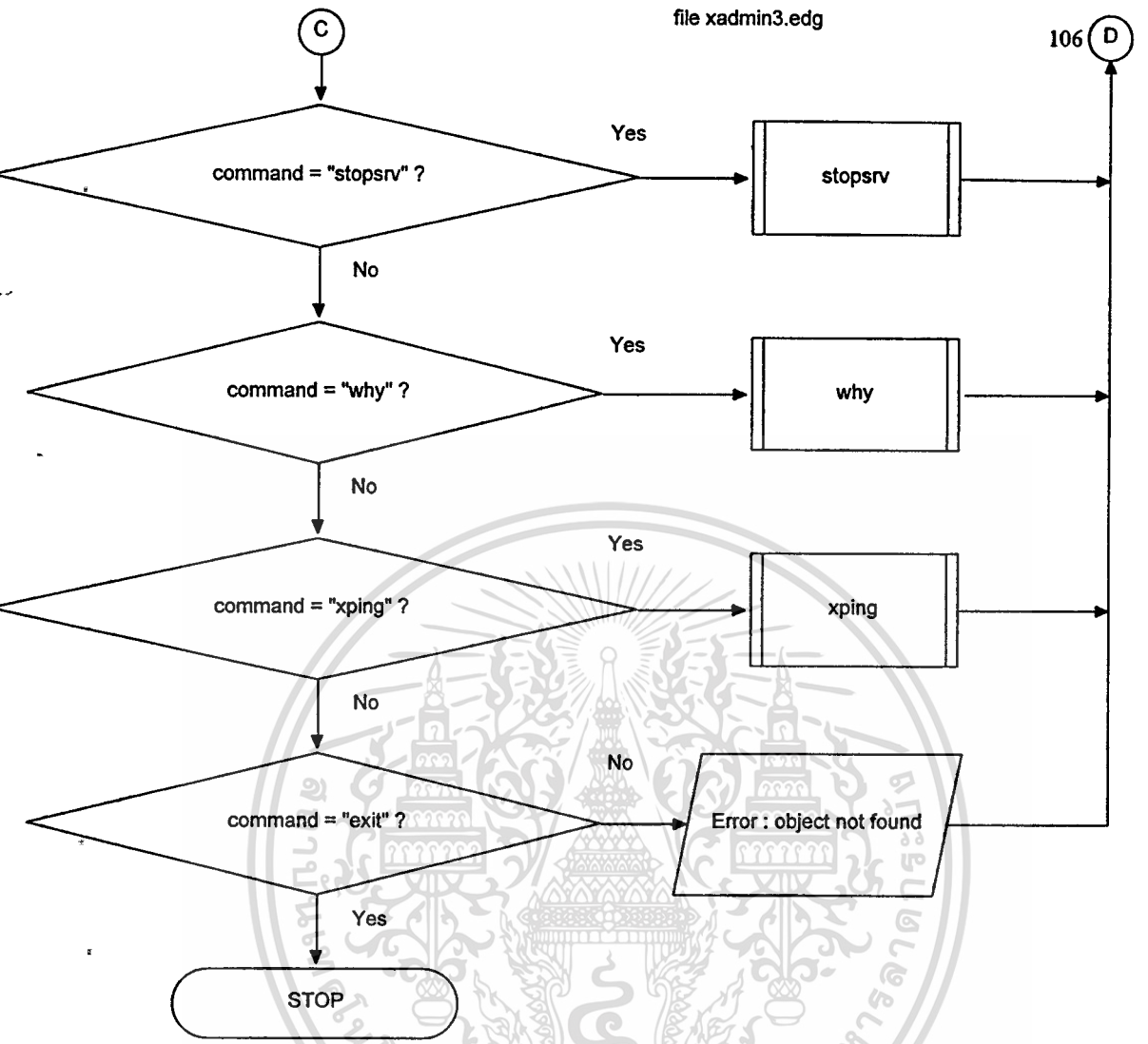


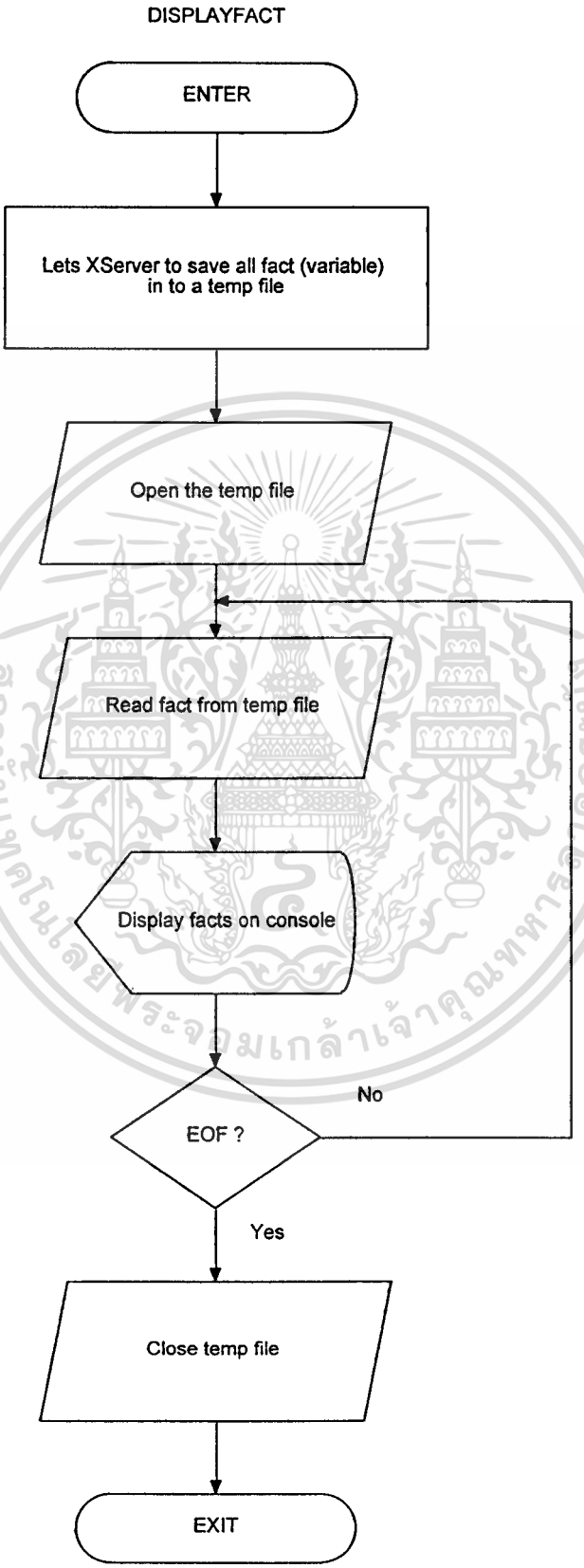
XADMIN: main 2/3

file xadmin2.edg

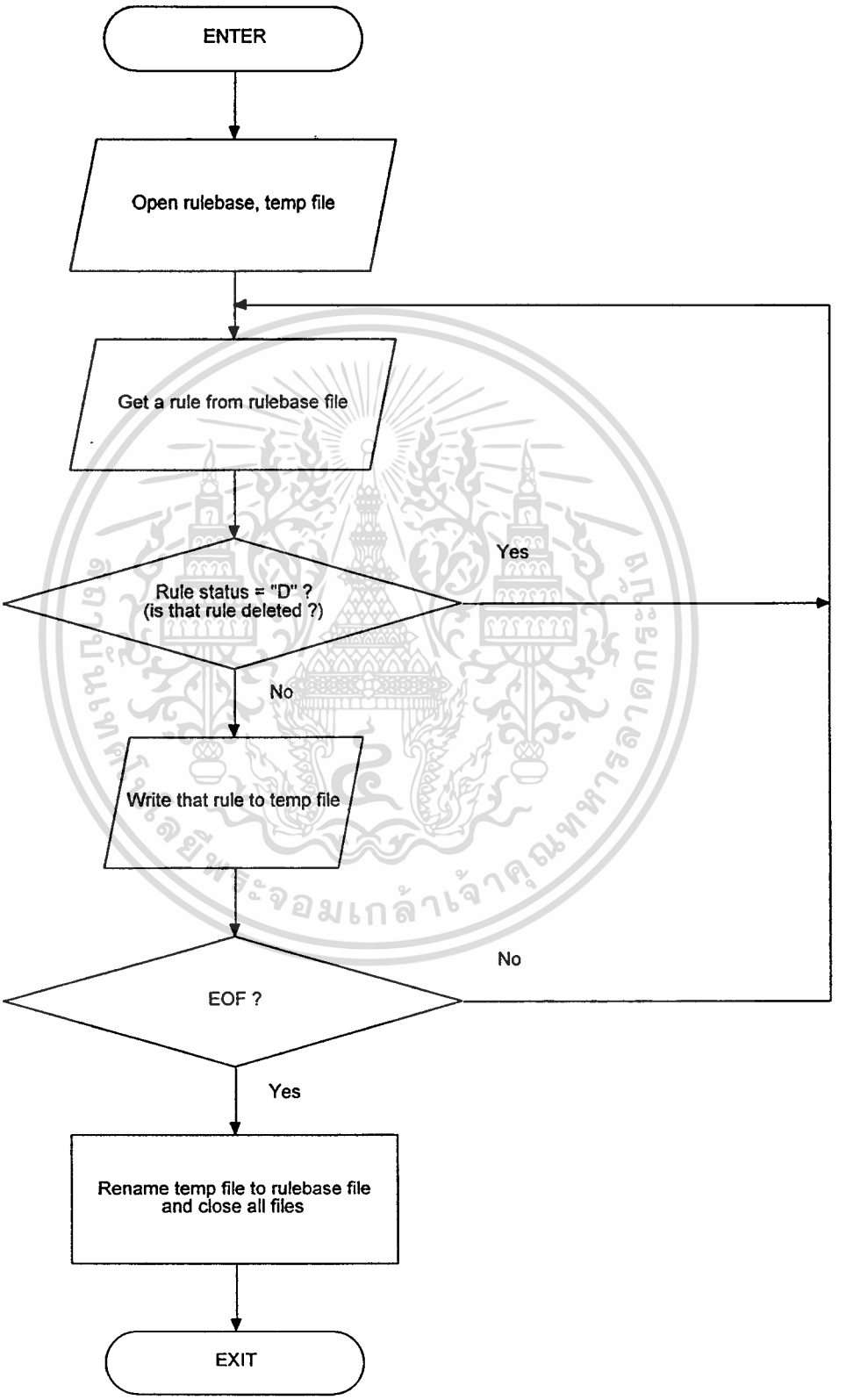


XADMIN: main 3/3





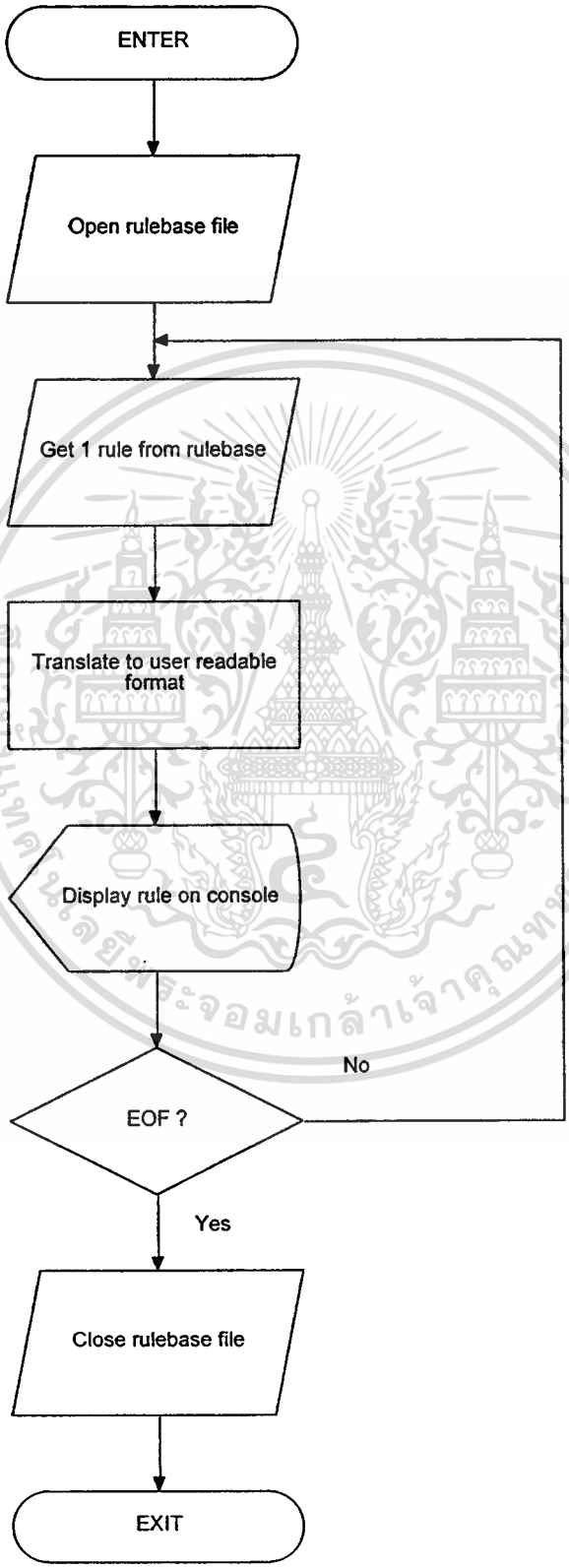
PACKRULE



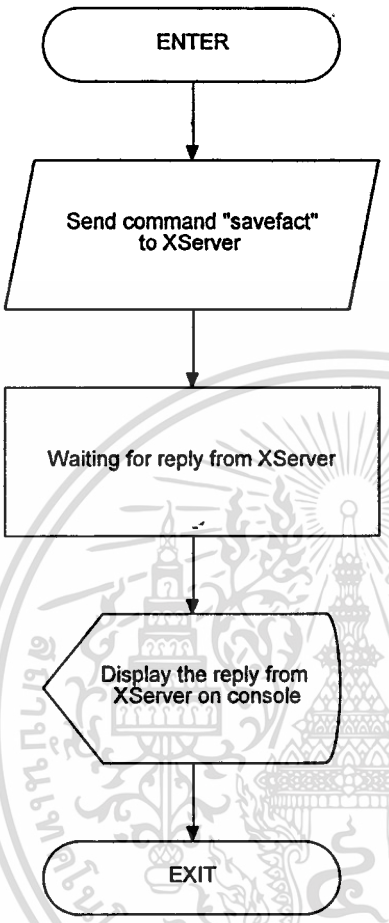
XADMIN : rulelist 1/1

file xadmin11.edg

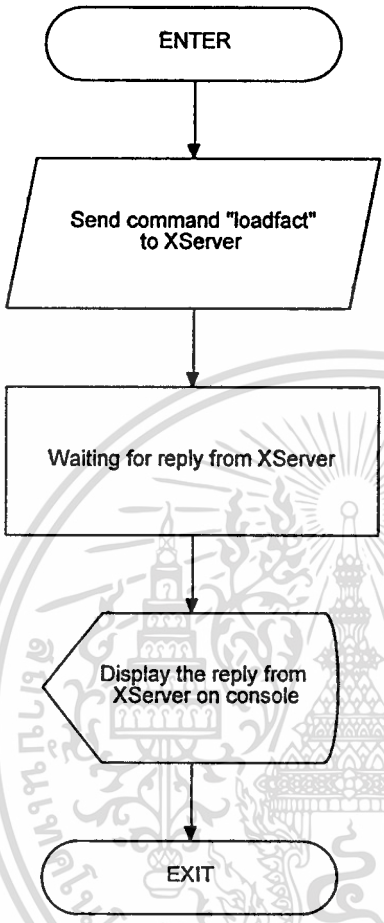
RULELIST

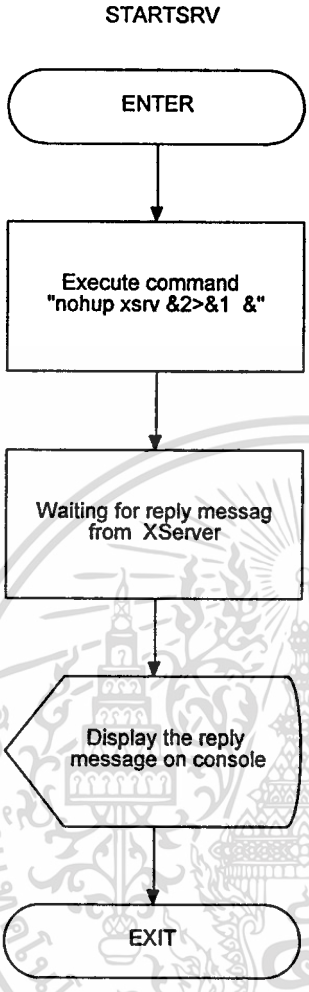


SAVEFACT

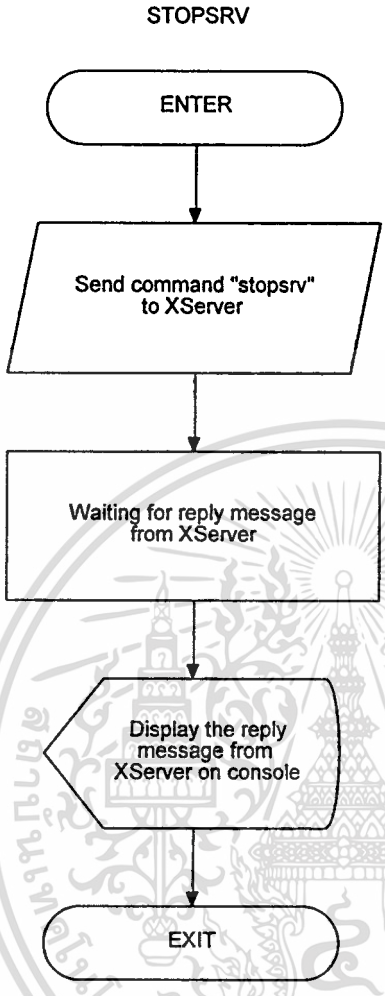


LOADFACT

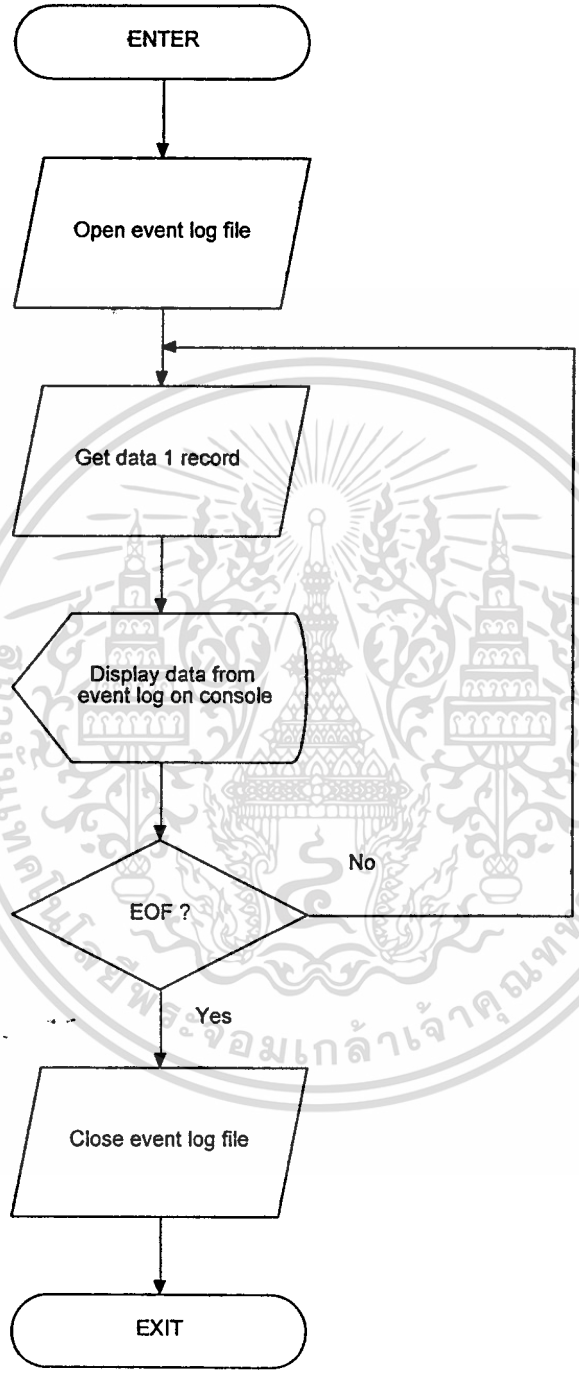




XADMIN : stopsrv 1/1

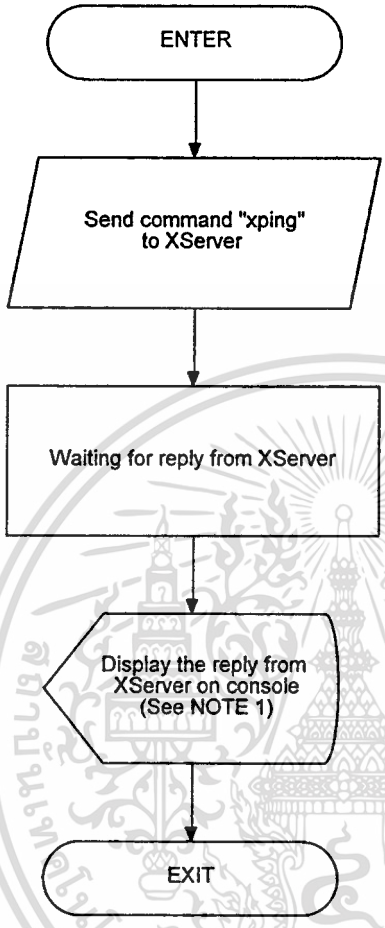


WHY (EXPLANATION FACILITY)



XADMIN : xping 1/1

XPING

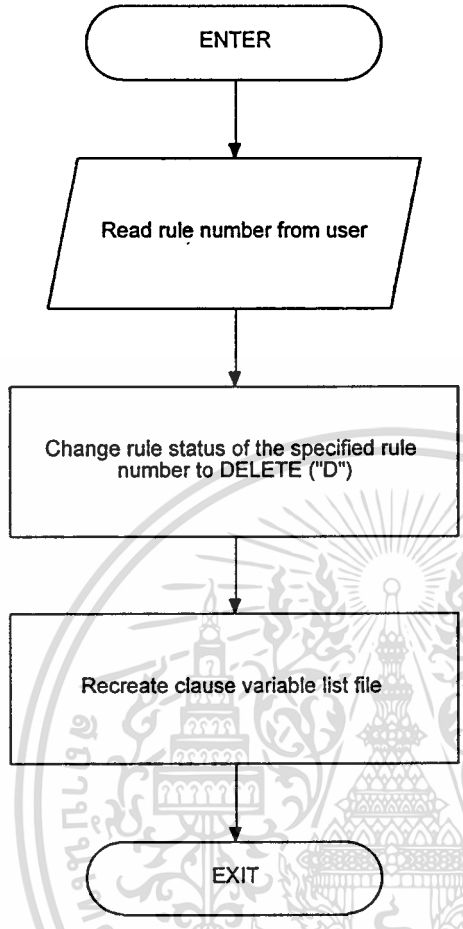


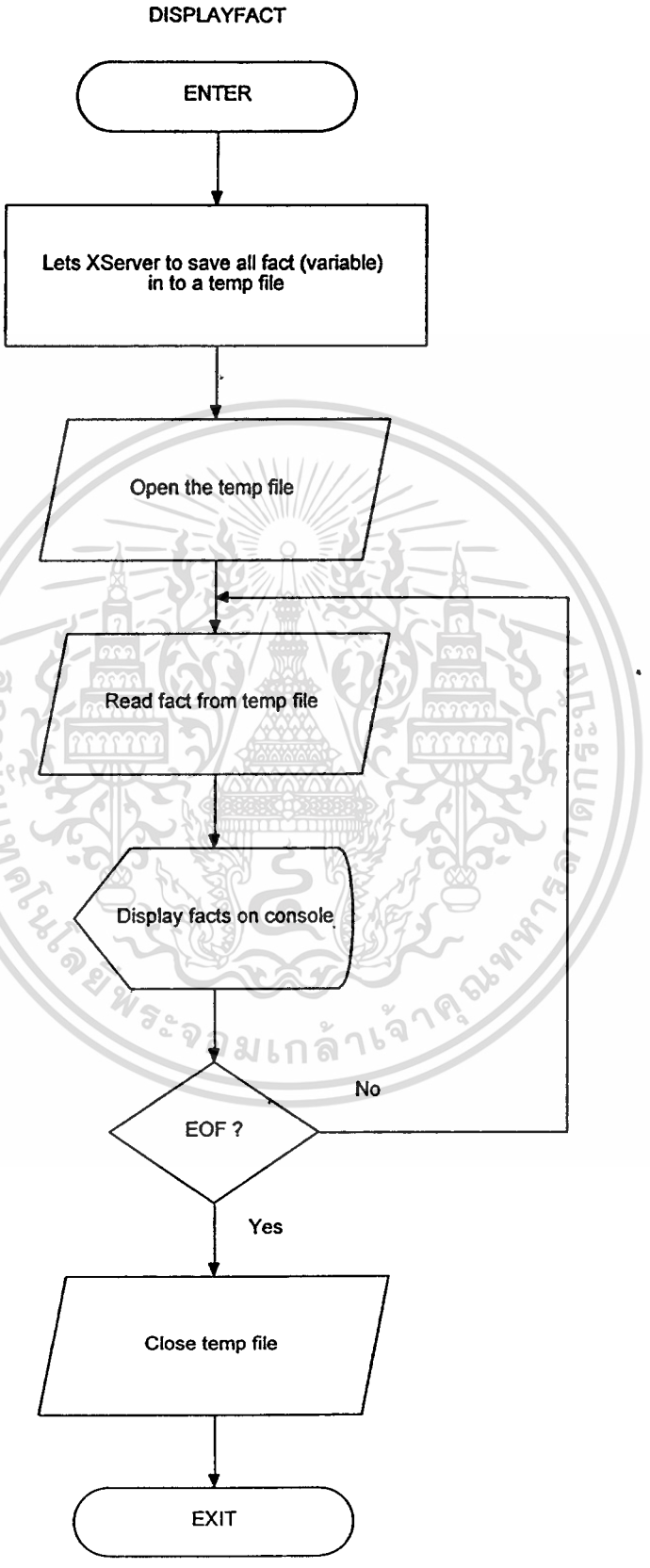
NOTE1:

Message = "XServ is alive" if XServer is up.
Message = "Cannot get Message queue" if XServer is not up.

file xadmin4.edg

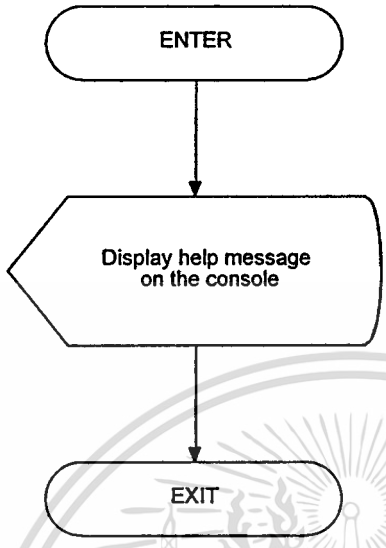
DELRULE

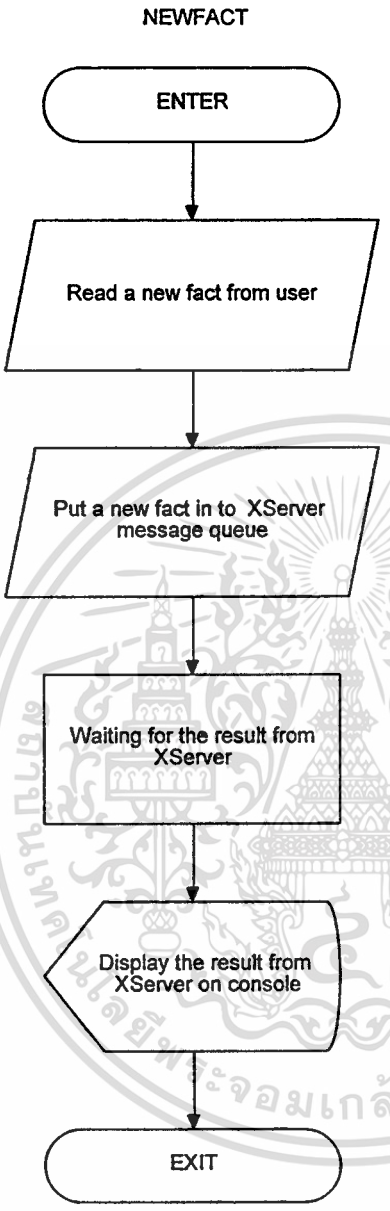




file xadmin6.edg

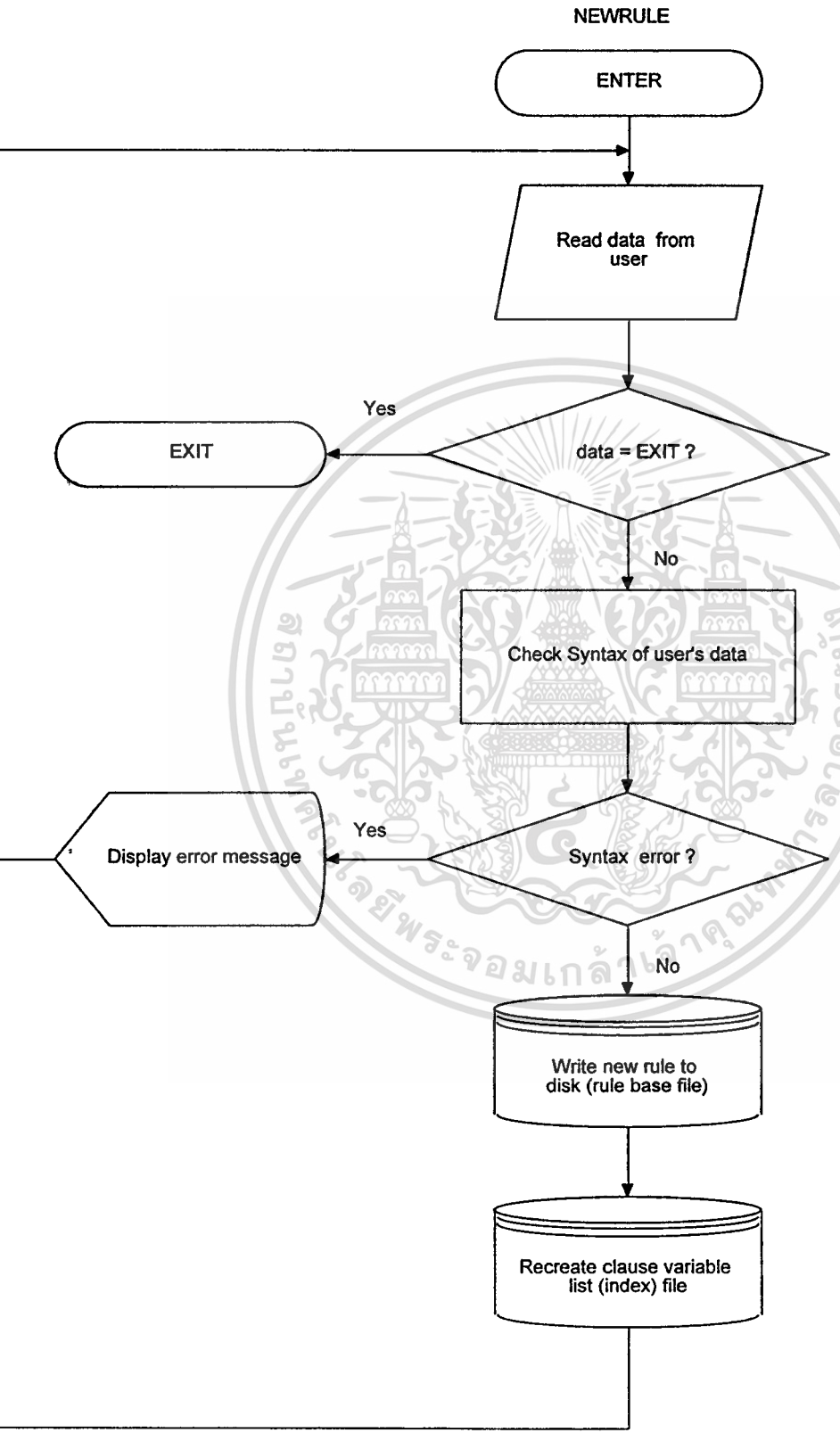
PRINTHELP





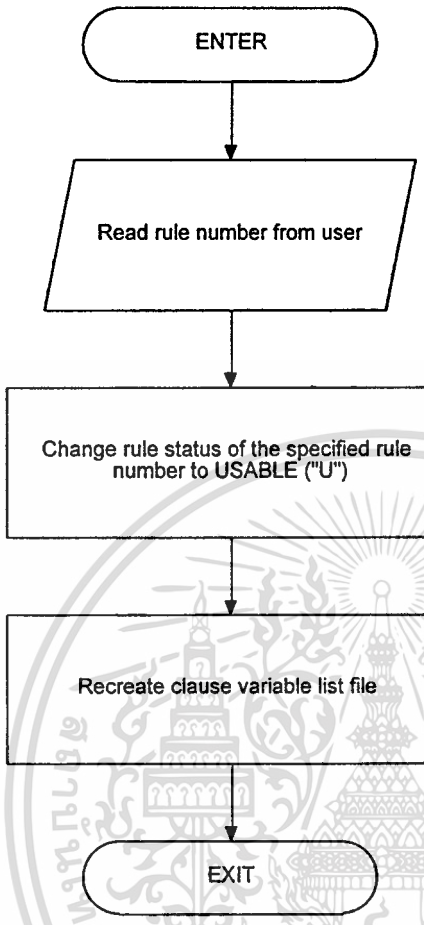
XADMIN : newrule 1/1

file xadmin8.edg



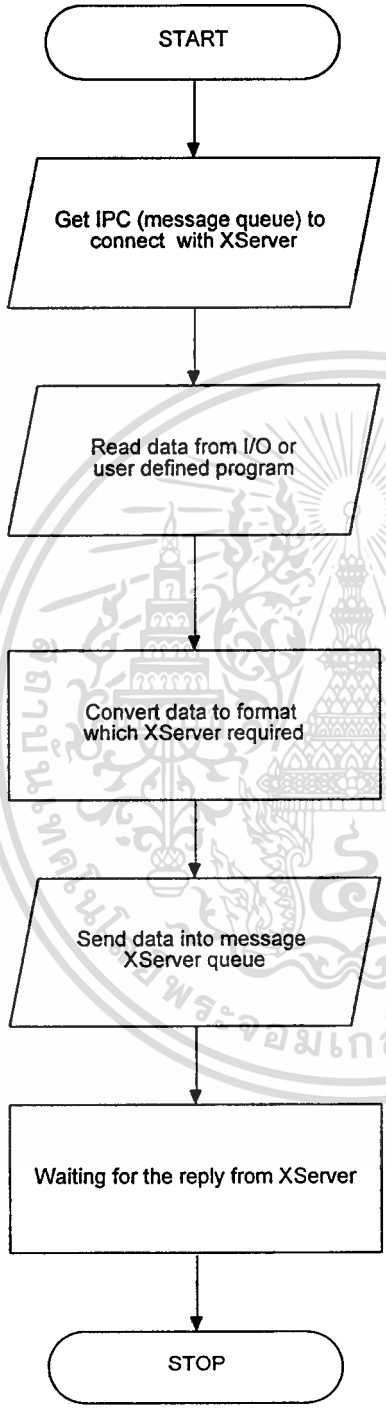
file xadmin9.edg

UDELRULE



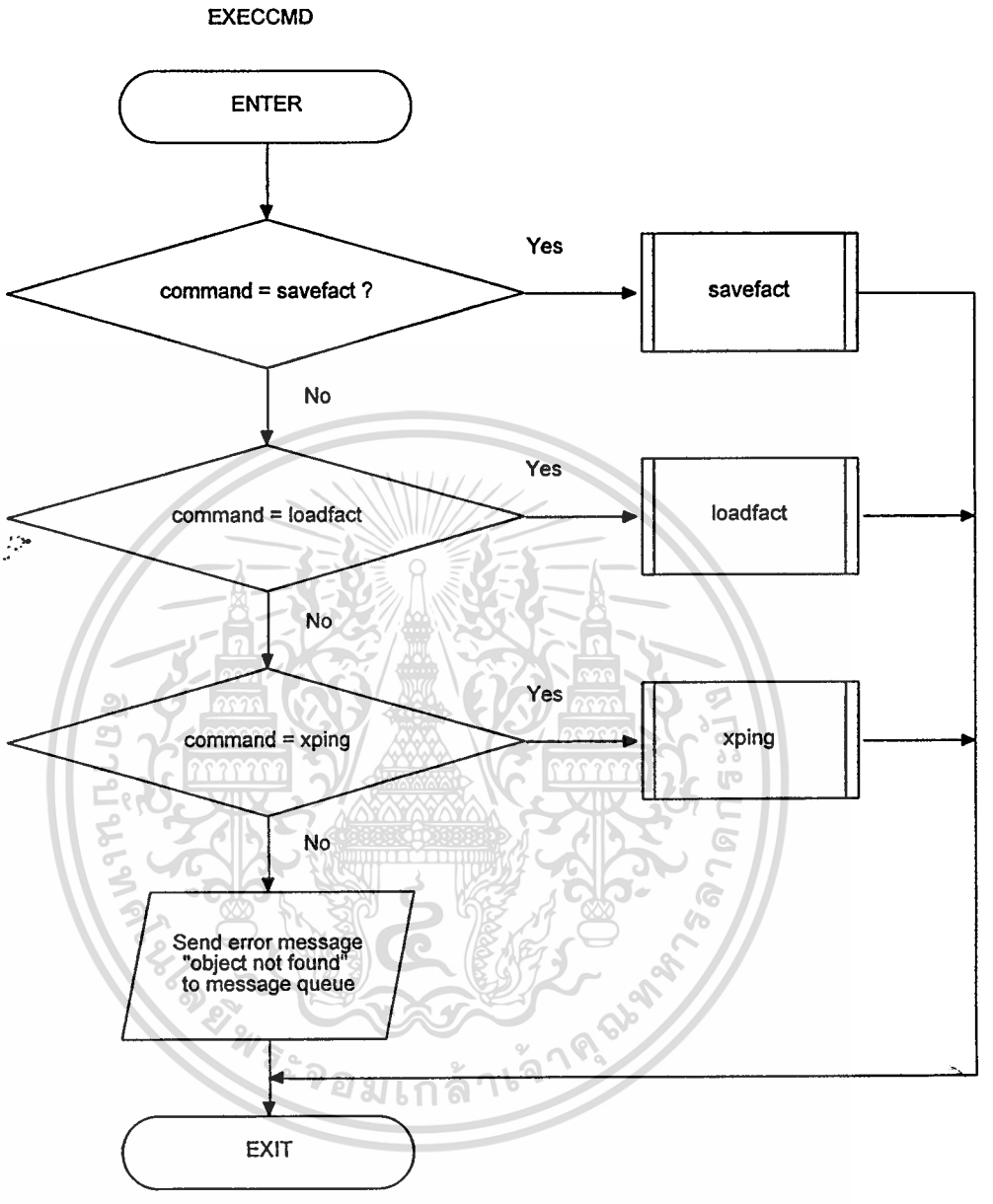
XClient : main 1/1

file xclient1.edg

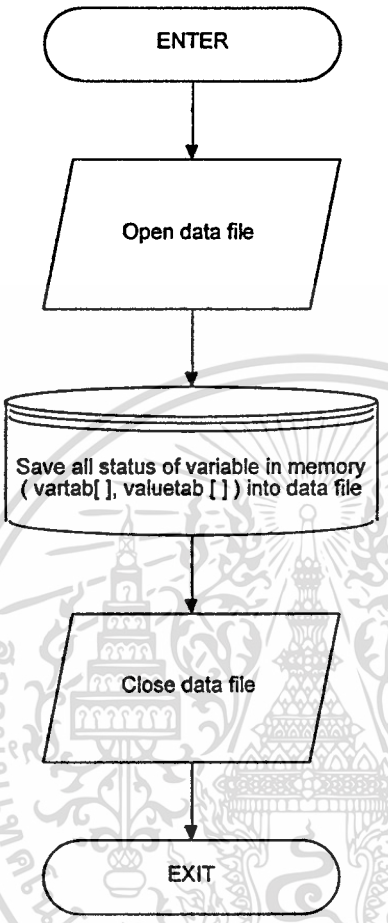


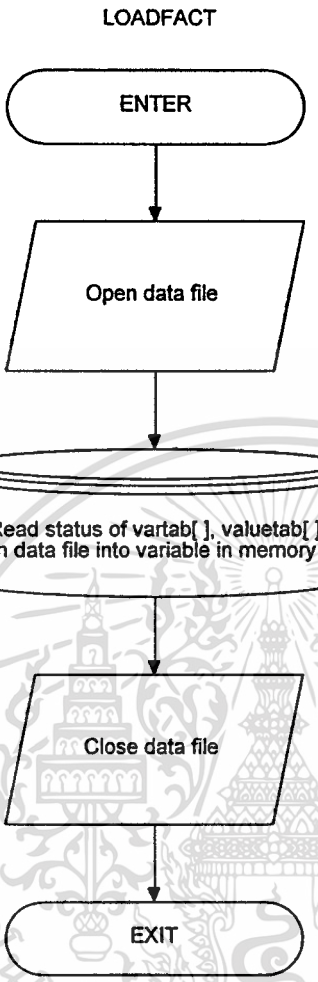
XServer : execcmd

file xserver2.edg

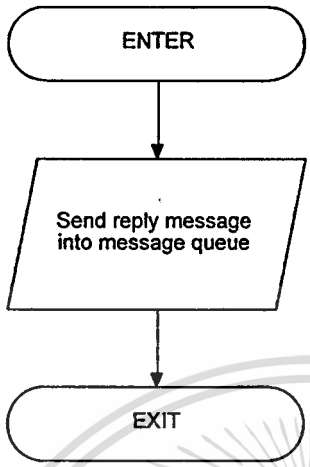


SAVEFACT



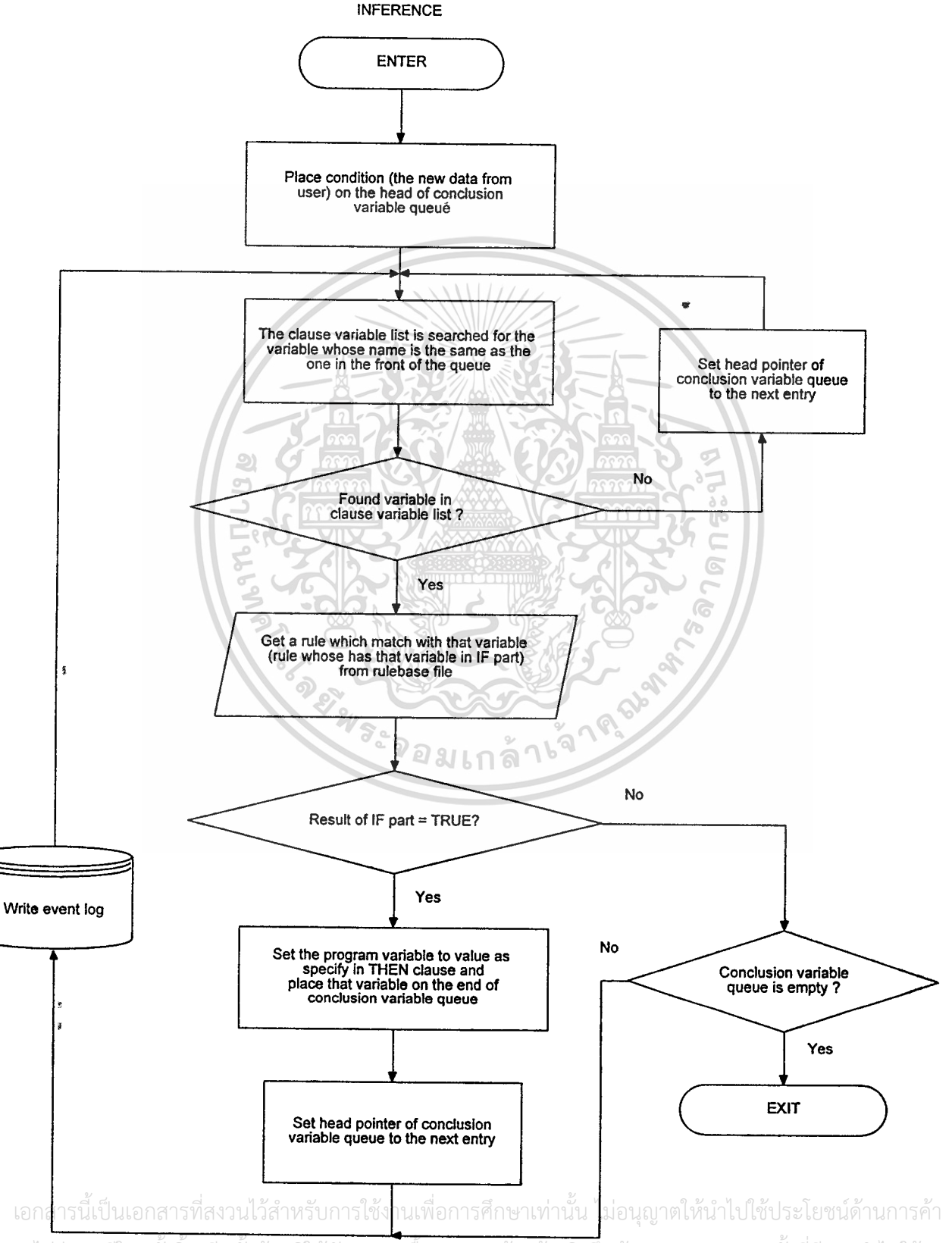


XPING



XServer : inference 1/1

file xserver6.edg



ภาคผนวก ข. โปรแกรม

Make file for XAdmin

```
xadmin : explan.o kaq.o io.o lex.o main.o message.o newfact.o search.o sort.o syntax.o seman2.o
then.o util.o
        cc -o xadmin explan.o kaq.o io.o lex.o main.o message.o newfact.o search.o sort.o
syntax.o seman2.o then.o util.o

explan.o  : explan.c
        cc -c -g explan.c
kaq.o     : kaq.c
        cc -c -g kaq.c
io.o      : io.c
        cc -c -g io.c
lex.o     : lex.c
        cc -c -g lex.c
main.o    : main.c
        cc -c -g main.c
message.o : message.c
        cc -c -g message.c
newfact.o : newfact.c
        cc -c -g newfact.c
search.o  : search.c
        cc -c -g search.c
sort.o    : sort.c
        cc -c -g sort.c
syntax.o  : syntax.c
        cc -c -g syntax.c
seman2.o  : seman2.c
        cc -c -g seman2.c
then.o    : then.c
```

```
cc -c -g then.c
util.o    : util.c
cc -c -g util.c
```

Make file for XServer

```
xsrv : explan.o lex.o message.o newfact.o search.o server3.o sort.o syntax.o seman2.o then.o util.o
xsrv.o
        cc -o xsrv explan.o lex.o message.o newfact.o search.o server3.o sort.o syntax.o
seman2.o then.o util.o xsrv.o

explan.o  : explan.c
        cc -c -g explan.c
lex.o     : lex.c
        cc -c -g lex.c
message.o : message.c
        cc -c -g message.c
newfact.o : newfact.c
        cc -c -g newfact.c
search.o  : search.c
        cc -c -g search.c
server3.o : server3.c
        cc -c -g server3.c
sort.o    : sort.c
        cc -c -g sort.c
syntax.o  : syntax.c
        cc -c -g syntax.c
seman2.o  : seman2.c
        cc -c -g seman2.c
then.o    : then.c
        cc -c -g then.c
util.o    : util.c
        cc -c -g util.c
```

```
xsrv.o    : xsrv.c
            cc -c -g xsrv.c
```

```
/* ---------------------------------------------------------------
  Program     Explanation sub-system
  Filename    explan.c
  By          M_Sukit
  Project     iesh.prj
  Created     94-11-02
  Modify      95-06-11
  Call
--------------------------------------------------------------- */
#include <stdio.h>

extern FILE *explanptr;

explanation()
{  char str[256];

  fseek(explanptr, 0, SEEK_SET);
  xprintf("\n");
  while(fgets(str, 255, explanptr) != NULL)
  {  if(strcmp(str, "No entry\n") == 0)  /* if there is no new fact to explan */
     {  xprintf("No new fact was entered\n");
           return(-1);
     }
     xprintf("%s", str);
  }
  xprintf("\n");
}

init_explan_file()
{ if((explanptr = fopen("explan.tmp","w")) == NULL)  /* open for clear content fo file */
```

```
  {  xprintf("Cannot open explan.tmp file\n");
     exit(1);
  }
  fprintf(explanptr, "No entry\n");
  fclose(explanptr);
  if((explanptr = fopen("explan.tmp","rb+")) == NULL)
  {  xprintf("Cannot open explan.tmp file\n");
     exit(1);
  }
}
```

```
/* ---------------------------------------------------------------
   Program     Knowledge Aquistition
   File name   kaq.c
   By          M_Sukit
   Project     the.prj
   Created     94-04-19
   Modify      94-11-02 -> change from menu to command line
                        -> change name of function knowledge_aq
                           to new_rule()
   Call        lexical_analyzer in lex.c
               syntax_analyzer  in syntax.c
   Contain     new_rule()
               delete_rule()
               undelete_rule()
               set_rule_attr()
               rule_list()
               pack_rule()
--------------------------------------------------------------- */

#include <stdio.h>
#include <string.h>
#include <ctype.h>   /*  for islower, isupper function */
#include <stdlib.h>  /*  for atoi() */
#include "lex.h"
#include "define.h"

extern long rule_position;
extern FILE *fptr;
extern int next_avail;
extern char *vartab[MAX_TABENTRY];
```

```
extern char *valuetab[MAX_TABENTRY];


void set_rule_attr(int num, char attr)
{  int i = 0;
   long fpos;                    /* position of current file pointer */
   char str[512];          /* tempory string          */


   if(num != -1)                 /* if num is number        */
   {   fseek(fptr, (long)0,SEEK_SET);
         while(i++ <=  num)
         {   fpos = ftell(fptr);
            fgets(str, 255, fptr);
         }
         fseek(fptr, fpos, SEEK_SET);  /* this line must not delete */
         fprintf(fptr,"%c", attr);      /* insert status mark */
   }
   fflush(fptr);
}
new_rule()
{  char istr[256];
   char lexstr[256];


   int         i=1;          /*  rule string pointer */
   int         ret;          /*  return value for function lex */


   xprintf("\n ENTER NEW RULE \n\n");
   xprintf("Please enter rule \n: ");
   gets(istr);
   while(strcmp(istr,"exit") != 0)
   {  ret = lexical_analyzer(istr, lexstr);
```

```
    xprintf("kaq1 : after lex\n");
    ret += syntax_analyzer(lexstr);
    xprintf("kaq2 : after syntax\n");
    ret += semantic_analyzer(lexstr, &i);
    xprintf("kaq3 : after seman\n");
    if(ret >= 0)
    {  rule_position = ftell(fptr); /*  find position of this rule */
        fseek(fptr, 0, SEEK_END);
        fprintf(fptr,"U%s",lexstr);  /* + is an rule status (see note.the) */
        puts(istr);              /*  print istr to console     */
        append_cvlfile(lexstr);     /*  update .cvl file          */
    }
    xprintf("\nPlease enter rule \n: ");
    gets(istr);
  }
  fflush(fptr);
}

delete_rule()
{  int num;
   char NUM[10];
   char str[512];

   xprintf("\n DELETE RULE \n\n");
   xprintf("Please enter number of rule : ");
   gets(NUM);
   num = atoi(NUM) - 1;
   set_rule_attr(num, 'D');
}
```

```
undelete_rule()
{   int num;
    char NUM[10];
    char str[512];

    xprintf("\n UNDELETE RULE \n\n");
    xprintf("Please enter number of rule : ");
    gets(NUM);
    num = atoi(NUM) - 1;
    set_rule_attr(num, 'U');
}

rule_list()
{    char rule[256];
     char otext[256];
     char attr;
     int   counter = 1;

     xprintf("\n RULE LIST \n\n");
     fseek(fptr, 0, SEEK_SET);
     while(fgets(rule, 255, fptr) != NULL)
     {  lex2text(rule,otext);
          attr = otext[0];       /* get attribute of rule   */
          otext[0] = ' ';        /* kill rule status sign   */
          xprintf("%5d).[%c]%s\n", counter++, attr, otext);
          if(counter == 23)
          {    xprintf("-- Pause, Hit any key to continue --\n");
          }
     }
}
```

```
pack_rule()
{  FILE *temprbase;
   FILE *cvlptr;
   char rule[256];
   int i;
   char name;
   char fullname[256];


   xprintf("\n PACK RULE \n\n");
   if((temprbase = fopen("temp", "wb")) == NULL)
   {  xprintf("Cannot open temp file, command aborted\n");
      return(-1);
   }
   fseek(fptr, 0, SEEK_SET);   /* rewind rbase file      */
   while(fgets(rule, 255, fptr) != NULL)
   {  if(rule[0] != 'D')       /* if rule is not deleted */
            {  fprintf(temprbase,"%s", rule);
       fprintf(stdout,"kaq:packrule:Rule % stored in tempfile\n",rule);
      }
   }
   fclose(temprbase);
   fclose(fptr);
   /*------- INITIALIZE VARTAB ---------*/
   for(i=0; i<next_avail ; i++)
      strcpy(vartab[i], "zzzzzzzz");
   i = 0;
   /*------ INITIALIZE VALUETAB --------*/
   for(i=0; i<next_avail ; i++)
```

```
    strcpy(valuetab[i], "null");
 i = 0;
 name = 'A';
 while((int)name <= (int)'Z')                /* repeat for file name a-z .cvl */
 {  sprintf(fullname,"%c.cvl",name);
    cvlptr = fopen(fullname, "w");      /* truncate all cvl file */
    fclose(cvlptr);
    name++;
 }
 printf("kaq:packrule check .cvl file, are them truncate?, hit any to cont\n");
 getchar();
 if((fptr = fopen("rbase.xsrv", "wb")) == NULL)  /* re-open for truncate rbase file */
 {  xprintf("Error when open rbase file, command aborted\n");
    exit(1);
 }
 if((temprbase = fopen("temp", "rb")) == NULL)  /* re-open for change to read mode */
 {  xprintf("Error when open temp file, command aborted\n");
    exit(1);
 }
 rule_position = 0;
 next_avail = 0;
 while((fgets(rule, 255, temprbase) != NULL))
 {  rule_position = ftell(fptr);
    fprintf(fptr,"%s",rule);  /*  'U' is an rule status (see note.the) */
    rule[0] = ' ';          /*  kill flag of rule        */
    printf("Kaq:packrule: Put rule %s to addto_cvlfile\n", rule);
    addto_cvlfile(rule);     /*  update .cvl file         */
 }
 fclose(fptr);
 fclose(temprbase);
```

```
  if((fptr = fopen("rbase.xsrv", "rb+")) == NULL)  /* bring fptr to the right mode */
  {  xprintf("Error when open rbase file, command aborted\n");
     exit(1);
  }

}
```

```
/* ------------------------------------------------------------
  Program     I/O Management
  File name   io.c
  By          M_Sukit
  Project     iesh.prj
  Created     94-11-02
  Modify      95-06-20
  Call
  Contained   startsrv()
              stopsrv()
------------------------------------------------------------ */
#include <stdio.h>
#include <stdlib.h>
#include  "srv.h"

start_srv()
{  xprintf("\n START EXPERT SYSTEM SERVER \n\n");
   system("sleep 2");   /* sleep because when stopsrv we sleep 3 sec*/
   system("nohup xsrv & > xsrv.out 2>&1");
   /* start by "system" command */
   system("sleep 1");   /* sleep for waiting from nohup message */
}

stop_srv()
{
    xprintf("\n STOP EXPERT SYSTEM SERVER \n\n");
    send2server("exit\n");
}

display_fact()
```

```
{   FILE *factfptr;
    char str[4096];

    xprintf("\n DISPLAY VALUE OF ALL FACT \n\n");
    send2server("displayfact\n");
    if((factfptr = fopen("tmpfact.xsrv", "r")) == NULL)
    {   xprintf("Cannot open tmpfact.xsrv file\n");
        return(-1);
    }
    while(fgets(str, 4090, factfptr) != NULL)
        xprintf("%s", str);
    fclose(factfptr);
}

fact2server()
{   char str[1024];   /* tempory string */

    xprintf("\nINPUT A FACT \n\n");
    xprintf("Please enter fact: ");
    fgets(str, 255, stdin);
    if(strcmp(str, "exit\n") == 0)
        return(0);
    send2server(str);
}
send2server(char *str)
{   int n;
    int id;    /* message queue id          */
    int pid;   /* process id of this process */

    if( (id = msgget(MKEY1, 0)) < 0)
```

```
{   xprintf("client: can't msgget message queue 1\n");
    return(-1);
}
pid = getpid();
sprintf(mesg.mesg_data, "%d|%s", pid, str);
n = strlen(mesg.mesg_data);
mesg.mesg_data[n] = '\0';
mesg.mesg_len = n;
mesg.mesg_type = 1L;
mesg_send(id, &mesg);
mesg.mesg_type = (long)pid;
if( (n = mesg_recv(id, &mesg)) >= 0)
{  mesg.mesg_data[mesg.mesg_len] = '\0';
   xprintf("send2server : OUTPUT\n%s", mesg.mesg_data);
}
else
   xprintf("send2server : data read error\n");
}
```

```
/* ---------------------------------------------------------------
  Program     Lexical Analyzer
  File name   lex.c
  By          M_Sukit
  Project     the.prj
  Created     94-05-21
  Modify      96-01-02
  Call
--------------------------------------------------------------- */
#include <stdio.h>
#include "lex.h"

lexical_analyzer(char *istr, char *lexstr)
{ int  i = 0;       /* counter for input string          */
  int  j = 0;       /* counter for output string         */

  while(istr[i] != '\0')
  { switch(istr[i])
    { case ('i') : if(istr[i+1] == 'f')
                   {  lexstr[j++] = IF;
                     i += 2;        /* skip 'if' */
                   }
                   else lexstr[j++] = istr[i++];
                   break;
      case ('t') : if(istr[i+1] == 'h' && istr[i+2] == 'e' && istr[i+3] == 'n')
                   {  lexstr[j++] = THEN;
                     i += 4;
                   }
              else lexstr[j++] = istr[i++];
                   break;
```

```
case ('e') : if(istr[i+1] == 'l' && istr[i+2] == 's' && istr[i+3] == 'e')
                { lexstr[j++] = ELSE:
                  i += 4;
                }
                else lexstr[j++] = istr[i++];
                break;
case ('c') : if(istr[i+1] == 'a' && istr[i+2] == 'l' && istr[i+3] == 'l')
                { lexstr[j++] = CALL;
                  i += 4;
                  while((istr[i] != ')') && (istr[i] != '\0'))
                        lexstr[j++] = istr[i++];
                }
        else lexstr[j++] = istr[i++];
                break;
case ('&') : if(istr[i+1] == '&')
                { lexstr[j++] = AND;
                  i += 2;
                }
                else lexstr[j++] = istr[i++];
                break;
case ('|') : if(istr[i+1] == '|')
                { lexstr[j++] = OR;
                  i += 2;
                }
                else i++;
                break;
case ('!') : if(istr[i+1] == '=')
                { lexstr[j++] = NEQ;
                  i += 2;
                }
```

```
                    break;
case ('=') : lexstr[j++] = EQ;
                    i += 1;
                    break;
case ('<') : if(istr[i+1] == '=')
                    {  lexstr[j++] = LTE;
                      i += 2;
                    }
                    else
                    {  lexstr[j++] = LT;
                      i += 1;
                    }
                    break;
case ('>') : if(istr[i+1] == '=')
                    {  lexstr[j++] = GTE;
                      i += 2;
                    }
                    else
                    {  lexstr[j++] = GT;
                      i += 1;
                    }
                    break;
case (' ') : i++;
                    break;
case ('\t'): i++;
                    break;
case ('\n'): i++;
                    break;
default    : lexstr[j++] = istr[i++];
                    break;
```

```
    }
  }
  lexstr[j]   = '\n';
  lexstr[j+1] = '\0';
  return(0);
}
```

```
/* ------------------------------------------------------------
  Program     Main program for my thesis
  File name   main.c
  By          M_Sukit
  Project     the.prj
  Created     94-05-21
  Modify      94-11-02 -> change from menu to command line
  Call
------------------------------------------------------------ */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "screen.h"
#include "define.h"

FILE *fptr;             /* Rule base file   */
FILE *explanptr;        /* Explanation file */
FILE *eventptr;         /* Event file       */
char istr[256];         /* Rule string      */
long rule_position;     /* position of rule in rbase  */
long rule_used_list[MAX_USED_LIST]; /* contain list of rules that already used in
                                       inference engine */


char *vartab[MAX_TABENTRY];
char *varq[MAX_Q_DEPT];  /* stack for keep var name for infer */
char *valuetab[MAX_TABENTRY];
char *valueq[MAX_Q_DEPT];       /* stack for keep value of varq  */
char replystr[4080];      /* Reply string for send output to client    */
```

```
int  next_avail = 0;      /*  Next available position in vartab        */
int  qtail = 0;           /*  tail pointer for fw_chainning queue      */
int  qhead = 0;           /*  head pointer for fw_chainning queue      */
int  last_used = 0;       /*  pointer of rule_use_list, specify the last
                                rule used */
int  lineno = 0;          /*  used in message.c */
main()
{ char teststr[256];    /* for debug */
  char command[256];      /* command line */

  xprintf("\nWell come to I/O Independent Expert System Shell\n\n");
  initialize();
  xprintf("[xAdmin] : ");
  fgets(command, 255, stdin);
  while(strcmp(command, "exit\n") != 0)              /* while not exit */
  {  lineno = 0;
    if(strcmp(command, "newrule\n") == 0)  new_rule();          /* OK */
     else
    if(strcmp(command, "delrule\n") == 0)  delete_rule();       /* OK */
     else
    if(strcmp(command, "rulelist\n")== 0)  rule_list();         /* OK */
     else
    if(strcmp(command, "why\n")    == 0)  explanation(); /* OK */
     else
    if(strcmp(command, "help\n")   == 0)  print_help();         /* OK */
     else
    if(strcmp(command, "udelrule\n")== 0)  undelete_rule();     /* OK */
     else
    if(strcmp(command, "packrule\n")== 0)  pack_rule();         /* OK */
     else
```

```
    if(strcmp(command, "testfact\n") == 0)  new_fact();          /* OK */
     else
    if(strcmp(command, "xping\n")   == 0)   send2server("xping\n");
     else
    if(strcmp(command, "newfact\n") == 0)   fact2server();       /* NoOK */
     else
    if(strcmp(command, "loadfact\n")== 0)  load_fact();         /* OK */
     else
    if(strcmp(command, "savefact\n")== 0)  save_fact();         /* OK */
     else
    if(strcmp(command, "startsrv\n")  == 0)  start_srv();       /* OK */
     else
    if(strcmp(command, "stopsrv\n")   == 0)  stop_srv();        /* OK */
     else
    if(strcmp(command, "displayfact\n") == 0) display_fact();   /* NoOk*/
     else
     { command[strlen(command) - 1] = '\0';
       xprintf("Object not found '%s'\n", command);
     }
    xprintf("[xAdmin] : ");
    fgets(command, 255, stdin);
  }
}


print_help()
{ xprintf("\n Help Subsystem  \n\n");
  xprintf(" delrule       : Mark a rule deleted                    \n");
  xprintf(" displayfact   : Print value of all fact                \n");
  xprintf(" help          : Provide this screen                    \n");
  xprintf(" help <command> : Explan syntax of the command                  \n");
```

```
  xprintf(" loadfact     : Load default facts from file          \n");
  xprintf(" newfact      : Enter a new fact                      \n");
  xprintf(" newrule      : Enter new rule                        \n");
  xprintf(" udelrule     : Undelete rule                         \n");
  xprintf(" packrule     : Delete all deleted mark rule          \n");
  xprintf(" rulelist     : List all rules                        \n");
  xprintf(" savefact     : Save default facts to file            \n");
  xprintf(" startsrv     : Start Expert System Server            \n");
  xprintf(" stopsrv      : Stop Expert System Server             \n");
  xprintf(" why          : Explan how system produce the advice  \n");
  xprintf(" xping        : Ping to XServer                       \n");
  xprintf(" exit         : Quit from this program                \n");
  xprintf("\n");
}


initialize()
{ FILE *cvlptr;
  char str[256];
  char oldstr[256] = "none";
  char name;          /*  File name not include . (a - z) */
  char fullname[8];   /*  File name include .             */
  int  i;

  xprintf("Initialize...\n");
  /*-------- ALLOCATE VARTAB ----------*/
  for(i=0; i<MAX_TABENTRY ; i++)
     vartab[i] = malloc(10);
  i = 0;
  /*------- INITIALIZE VARTAB ---------*/
```

```
for(i=0; i<MAX_TABENTRY ; i++)
   strcpy(vartab[i], "zzzzzzzz");
i = 0;
/*------- ALLOCATE VALUETAB ---------*/
for(i=0; i<MAX_TABENTRY ; i++)
   valuetab[i] = malloc(10);
i = 0;
/*------ INITIALIZE VALUETAB --------*/
for(i=0; i<MAX_TABENTRY ; i++)
   strcpy(valuetab[i], "null");
i = 0;
/*-------- ALLOCATE VARQUEUE---------*/
for(i=0; i<MAX_Q_DEPT ; i++)
   varq[i] = malloc(9);
i = 0;
/*------- ALLOCATE VALUEQUEUE--------*/
for(i=0; i<MAX_Q_DEPT ; i++)
   valueq[i] = malloc(9);
i = 0;
/*--- INITIALIZE RULE USED LIST -----*/
for(i=0; i<MAX_USED_LIST ; i++)
   rule_used_list[i] = -1;
i = 0;
/*------ OPEN RBASE FILE --------*/
if((fptr = fopen("rbase.xsrv","rb+")) == NULL)
{  fptr = fopen("rbase.xsrv","w");   /* if rbase file not exist then */
   fclose(fptr);                     /* create it                    */
   if((fptr = fopen("rbase.xsrv","rb+")) == NULL)
   {  xprintf("Cannot open rbase.xsrv file\n");
         exit(1);
```

```
    }
}
fseek(fptr,0,SEEK_END);
rule_position = ftell(fptr);
/*------ OPEN EXPLANATION FILE --------*/
init_explan_file();

/*------ OPEN EVENT FILE --------*/
if((eventptr = fopen("event.log","a")) == NULL)  /* open for clear content fo file */
{   xprintf("Cannot open event.log file\n");
    exit(1);
}

/*------- OPEN CVL FILE ---------*/
name = 'A';
while((int)name <= (int)'Z')                /* repeat for file name a-z .cvl */
{  sprintf(fullname,"%c.cvl",name);
   if((cvlptr = fopen(fullname, "r+")) != NULL)    /* if ?.cvl is exist */
   {   xprintf("Processing [%s] \n",fullname);
          sort_cvlfile(fullname);
          while(fgets(str, 255, cvlptr) != NULL)
          {  if(strcmp(oldstr, str) != 0)
             {  strcpy(vartab[i], str);
                next_avail++;
                i++;
             }
             strcpy(oldstr, str);
             fgets(str, 255, cvlptr);      /*  Skip position of this var */
          }
   }
```

```
    fclose(cvlptr);
    name++;
  }
  xprintf("Initialize complete\n");
}
```

```
/* ---------------------------------------------------------------
   Program     Message management
   File name   message.c
   By          M_Sukit
   Project     the.prj
   Created     94-05-21
   Modify      94-11-02 -> change from menu to command line
   Call
---------------------------------------------------------------- */

#include <stdio.h>
#include <stdarg.h>

extern int lineno;

err(char *format, ...)
{   va_list argptr;

    fprintf(stdout , "Error: ");
    va_start(argptr, format);
    vprintf(format, argptr);
    va_end(argptr);
}
xprintf(char *format, ...)
{   va_list argptr;
    if(lineno == 15)
    { fprintf(stdout, "-- HIT ENTER TO CONTINUE -- \r");
      getchar();
      fprintf(stdout, "                          \r");
      lineno = 0;
```

```
    }
    va_start(argptr, format);
    vprintf(format, argptr);
    va_end(argptr);
    lineno++;
}
```

```
/* ---------------------------------------------------------------
  Program    Evaluate rule
  File name  newfact.c
  By         M_Sukit
  Project    the.prj
  Created    94-05-29
  Modify
  Call
  Contain    new_fact()
             do_new_fact()
             evaluate_rule()
             convert_input()
             check_if()
             update_vartab_entry()
           save_fact()
             load_fact()
--------------------------------------------------------------- */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include "lex.h"
#include "define.h"

extern FILE *fptr;
extern FILE *explanptr;
extern FILE *eventptr;
extern char *vartab[MAX_TABENTRY];
extern char *varq[MAX_Q_DEPT];
```

```
extern char *valuetab[MAX_TABENTRY];
extern char *valueq[MAX_Q_DEPT];
extern char replystr[4080];
extern long rule_used_list[MAX_USED_LIST];
extern int  next_avail;
extern int  qtail;
extern int  qhead;
extern int  last_used;

FILE *cvlptr;

new_fact()
{  char str[256];

   fseek(explanptr, 0, SEEK_SET);  /* rewind explanation file */
   xprintf("\nINPUT A FACT\n");
   xprintf("Please enter fact: ");
   fgets(str, 255, stdin);
   do_new_fact(str);
}

do_new_fact(char *str)
{  char VAR[256];
   char OPER[256];
   char lexstr[256];
   char VALUE[256];
   int      result;
   int      loop = 0; /* count for maximum inference loop */

   fprintf(eventptr,"-> [newfact] %s", str);
```

```
  fflush(eventptr);
  xprintf("newfact: FACT=[%s]", str);
  lexical_analyzer(str, lexstr);
  xprintf("newfact: before convert_input\n");
  convert_input(lexstr, VAR, OPER, VALUE);
  update_vartab_entry(VAR, VALUE);    /*  change VAR to VALUE */
  xprintf("newfact: before init_explan_file\n");
  init_explan_file();                 /*  truncate file      */
  fprintf(explanptr, "The latest fact is %s", str);
  fflush(explanptr);
  result = evaluate_rule(VAR);
  while((qhead != qtail) && (loop < 256)) /* no var in stack and eva time < max loop */
  {
    xprintf("newfact: before evaluate_rule\n");
    result = evaluate_rule(varq[qhead]);
    xprintf("newfact: after evaluate_rule\n");
    qhead++;
    if(qhead == MAX_Q_DEPT)   /* circular queue */
          qhead = 0;
    loop++;
    xprintf("newfact: before end while loop, qhead =%d, tail = %d, loop= %d \n",qhead, qtail,
loop);
  }
  xprintf("newfact: after end while loop\n");
  for(loop=0; loop<MAX_USED_LIST ;loop++)
    rule_used_list[loop] = -1;
  xprintf("newfact: before end of function\n");
  return(result);
}
```

```
evaluate_rule(char *VAR)
{  char cvlfname[10];   /* cvl file name                  */
   char variable[256];  /* variable in cvl file           */
   char var_pos[20];    /* position of variable in rbase file  */
   char rule[256];      /* rule that get from rbase       */.
   char        *p;  /* return value of fgets,if NULL -O1> EOF*/
   char        *q;  /* return value of fgets,if NULL -> EOF*/
   int         result;
   int         i;

   sprintf(cvlfname,"%c.cvl",VAR[0]);
   xprintf("newfact:Test 0 : cvlfname=%s\n", cvlfname);
   if((cvlptr = fopen(cvlfname,"r")) == NULL)
   {  xprintf("newfact:Test 1.1 : \n");
   VAR[strlen(VAR)-1] = '\0';
         fprintf(explanptr, "Variable [%s] is not in if clause of any rule, so inference stop\n",
VAR);
   fflush(explanptr);
         VAR[strlen(VAR)+1] = '\0';
         return(1);
   }
   xprintf("newfact:Test 1.2 : \n");
   xprintf("newfact:Test 1.3 : \n");
   while((fgets(variable, 256, cvlptr) != NULL) && (fgets(var_pos, 256, cvlptr) != NULL))
   {  var_pos[strlen(var_pos)-1] = '\0';  /* kill \n at the end of string */
      xprintf("newfact:Test 4 : variable=%s, VAR=%s\n", variable, VAR);
         if((strcmp(variable, VAR) == 0) && (!rule_has_used(var_pos)))
         {  i = 2;        /* init pointer for rule (skip status and 'if'*/
            fseek(fptr, atol(var_pos), SEEK_SET);
```

```
        xprintf("newfact:Test 5 : variable=%s, VAR=%s, Varpost = %s\n", variable, VAR,
var_pos);
            fgets(rule, 256, fptr);
        xprintf("newfact:Test 1: rule=%s\n", rule);
            if(rule[0] == 'U')  /* if this rule is usable */
            {
          xprintf("newfact:Test 6 this rule is usable\n");
              result = check_if(rule, &i);
              i++;          /* skip THEN character   */
              if(result == 1)  /* if IF_CLAUSE is TRUE then do THEN_CLAUSE */
              {  rule_used_list[last_used] = atol(var_pos); /* keep track of used rule */
                    last_used++;
             xprintf("newfact:Test 7: rule=%s\n", rule);
                    do_then(rule, &i);
             xprintf("newfact:Test 8: rule=%s\n", rule);
              }
            }
     xprintf("newfact:Test 9: rule=%s\n", rule);
         }
   }
   fclose(cvlptr);
}

convert_input(char str[256], char VAR[256], char OPER[256],
              char VALUE[256])
{  int i = 0;
   int varlength;

   if(isvar(str, i))
   {  getvar(VAR, str, &i);
```

```
        varlength = strlen(VAR);
        VAR[varlength]   = '\n';
        VAR[varlength+1] = '\0';
        if(isoper(str, i))
        {   OPER[0] = str[i];
            OPER[1] = '\0';
            i++;
            if(isvalue(str, i))
              getvalue(VALUE, str, &i);
            else xprintf("[CON] : After operator should be numberic value\n");
        }
        else xprintf("[CON] : After variable should be operator\n");
    }
}

int check_if(char *str, int *i)
{   char VAR[10];
    char OPER;
    char VALUE[10];
    int  result = 0;
    int  value, var_value;
    int  var_pointer;          /* pointer to value VAR in vartab */

    xprintf("newfact.check_if : Test 1\n");
    if(isvar(str, *i))
    {   getvar(VAR, str, i);
        strcat(VAR,"\n");
      xprintf("newfact.check_if : VAR=%s\n", VAR);
        if(isoper(str, *i))
        {   OPER = str[*i];
```

```
        (*i)++;
        if(isvalue(str, *i))
        {  getvalue(VALUE, str, i);
              strcat(VALUE, "\n");
              var_pointer = search_var_table(VAR);
       xprintf("newfact.check_if : VAR=%s OP=%c VALUE=%s\n", VAR, OPER, VALUE);
              result = calculate(valuetab[var_pointer], OPER, VALUE);
       xprintf("newfact.check_if : result=%d\n", result);
              while(islogic(str, *i))
              {  OPER = str[*i];
                 (*i)++;          /* Skip operator */
                 result = calculate(result, OPER, check_if(str, i) );
              }
              return(result);
        }
        else xprintf("[CIF] : After operator should be numberic value\n");
     }
     else xprintf("[CIF] : After variable should be operator\n");
}
else
if(str[*i] == '(')
{  (*i)++;
     result = check_if(str, i);
     if(str[*i] != ')')
     {  xprintf("[CIF] : Missing ')'\n");
        return(-1);
     }
     else (*i)++;       /* Skip ')' */
     while(islogic(str, *i))
```

```
        {  OPER = str[*i];
           (*i)++;          /* Skip operator */
           result = calculate((char *)result, OPER, (char *)check_if(str, i) );
        }
    }
    return(result);
}
/*-----------------------------------------------------------------------
 Function : update_vartab_entry
 For      : update variable VAR in vartab to value = VALUE
 Input    : VAR -> variable name
            VALUE -> new variable value
 Output   : 0 = Seccess
            -1 = Error
-----------------------------------------------------------------------*/

update_vartab_entry(char *VAR, char *VALUE)
{   int var_pointer;

    var_pointer = search_var_table(VAR);
    if(var_pointer != -1)              /*  if VAR exist in vartab  */
        strcpy(valuetab[var_pointer], VALUE); /*  then update it        */
    else                               /*  else add one          */
    {   strcpy(vartab[next_avail], VAR);
        strcpy(valuetab[next_avail], VALUE);
    }

}

save_fact()
```

```
{  FILE *factfptr;                    /* domain database file */
   int i = 0;
   char vartmp[9];
   char valuetmp[9];

   testvar_value("save_fact", 10);
   fprintf(eventptr, "-> [savefact]\n");
   fflush(eventptr);
   if((factfptr = fopen("fact.xsrv", "w")) == NULL)
   {   xprintf("Cannot open fact.xsrv file, cannot save fact\n");
      return(-1);
   }
   while((strcmp(vartab[i], "zzzzzzzz") != 0)&&(i < MAX_TABENTRY)) /* while not end of data
*/
   {   strcpy(vartmp, vartab[i]);
      strcpy(valuetmp, valuetab[i]);
      vartmp[strlen(vartab[i]) - 1] = '\0';          /* kill \n */
      if(strcmp(valuetmp, "null") != 0)
            valuetmp[strlen(valuetab[i]) - 1] = '\0';   /* kill \n */
      fprintf(factfptr, "%s=%s\n", vartmp, valuetmp);
      i++;
   }
   fclose(factfptr);
   strcpy(replystr, "Saved\n");
   xprintf("Saved\n");
}

save_fact_to_tmp_file()  /* use for display fact function */
{  FILE *factfptr;      /* domain database file */
   int i = 0;
```

```
  char vartmp[9];
  char valuetmp[9];

  xprintf("newfact. tmpfact : Begin\n");
  testvar_value("save_fact", 10);
  fprintf(eventptr, "-> [displayfact]\n");
  fflush(eventptr);
  if((factfptr = fopen("tmpfact.xsrv", "w")) == NULL)
  {  xprintf("newfact. tmpfact : Cannot open tmpfact.xsrv file, cannot save fact\n");
     return(-1);
  }
  while((strcmp(vartab[i], "zzzzzzzz") != 0)&&(i < MAX_TABENTRY)) /* while not end of data
*/
  {  strcpy(vartmp, vartab[i]);
     strcpy(valuetmp, valuetab[i]);
     vartmp[strlen(vartab[i]) - 1] = '\0';        /* kill \n */
     if(strcmp(valuetmp, "null") != 0)
           valuetmp[strlen(valuetab[i]) - 1] = '\0';  /* kill \n */
     fprintf(factfptr, "%s=%s\n", vartmp, valuetmp);
     i++;
  }
  fclose(factfptr);
  strcpy(replystr, "Temp saved\n");
  xprintf("Temp saved\n");
}

load_fact()
{  FILE *factfptr;                    /* domain database file */
  int i = 0;
  char str[MAX_TABENTRY];
```

```
  char lexstr[MAX_TABENTRY];
  char VAR[256];
  char OPER[256];
  char VALUE[256];

  fprintf(eventptr, "-> [loadfact]\n");
  fflush(eventptr);
  if((factfptr = fopen("fact.xsrv", "r")) == NULL)
  {  xprintf("Cannot open fact.xsrv file\n");
     return(-1);
  }
  while((fgets(str, MAX_TABENTRY-1, factfptr) != NULL)&&(i < MAX_TABENTRY)) /*
while not end of data */
  {  lexical_analyzer(str, lexstr);
     convert_input(lexstr, VAR, OPER, VALUE);
     update_vartab_entry(VAR, VALUE);    /*  change VAR to VALUE */
  }
  fclose(factfptr);
  strcpy(replystr, "Loaded\n");
  xprintf("Loaded\n");
}
testvar_value(char *str, int max)
{  int i;

  xprintf("TESTVAR.%s :", str);
  for (i=0;i<10;i++)
     xprintf("var%s=%s\n", vartab[i], valuetab[i]);
}
```

```
/* ---------------------------------------------------------------
  Program    Search value of variable in variable table
  File name  search.c
  By         M_Sukit
  Project    the.prj
  Created    94-06-12
  Modify
  Call
  Contain    search_var_table(char *VAR)
             insert_new_var(char *var)
--------------------------------------------------------------- */

#include <stdio.h>
#include <string.h>

#include "lex.h"

#define  MAX_TABENTRY 100

extern char *vartab[MAX_TABENTRY];
extern char *valuetab[MAX_TABENTRY];
extern int  next_avail;

/*-----------------------------------
 Function : search
 For      : find value of variable
 input    : char *var
 output   : int var_value
 modify   :
 -----------------------------------*/
```

```
search_var_table(char *VAR)
{   int i = 0;

    while(i < MAX_TABENTRY)
    {  if(strcmp(VAR, vartab[i]) == 0)
            return(i);
         else
           i++;
    }
    return(-1);   /*  No variable named VAR in vartab  */
}

insert_newvar(char *var)
{  int i = 0;
   int j = 0;
   char tempvar[9];

   while( (i < MAX_TABENTRY) && i <= next_avail )
   {   if(strcmp(vartab[i], var) == 0)  /* if this variable already exist */
            return(0);
         if(strcmp(vartab[i], var) > 0)   /* if vartab[i] > var  */
         {  j = next_avail;
            next_avail++;
            while(j > i)
            {  strcpy(vartab[j], vartab[j-1]);
                  strcpy(valuetab[j], valuetab[j-1]);
                  j--;
            }
            strcpy(vartab[i], var);
            strcpy(valuetab[i], "null");
```

```
            return(0);
        }
        i++;
    }
    if(i != MAX_TABENTRY)
    {   strcpy(vartab[i-1], var);
        strcpy(valuetab[i-1], "BLANK");
        return(0);
    }
    else
    {   err("[INS] : Number of variable > MAX_TABENTRY\n");
        return(-1);
    }
}
```

```
/* ---------------------------------------------------------------
  Program    Create message queue for server
  File name  server3.c
  By         M_Sukit
  Project    iesh.prj
  Created    94-12-05
  Modify
  Call
--------------------------------------------------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "msgq.h"
#include "srv.h"

Mesg mesg;

extern char replystr[4080];
do_exp_server()
{  int id;       /* message queue id */
   if( (id = msgget(MKEY1, PERMS | IPC_CREAT)) < 0 )
     xprintf("Server : can't get message queue 1");
   server(id);
   xprintf("Process finish\n");
   exit(0);
}
server(id)
int id;
{  int   n, filefd;
```

```
int  clientpid;
char  errmesg[256];
char  strinput[1024];

mesg.mesg_type = 1L;
if( (n = mesg_recv(id, &mesg)) <= 0)
    xprintf("Server : Input read error\n");
mesg.mesg_data[n] = '\0';
xprintf("server3:mes.data=%s", mesg.mesg_data);
testvar_value("server3", 10);
xprintf("server3:Test point 1\n");
get_client_data(mesg.mesg_data, &clientpid, strinput);
while(strcmp(strinput, "exit\n") != 0)
{  strcpy(replystr, "");
   if(strcmp(strinput, "savefact\n") == 0)
     save_fact();
   else
   if(strcmp(strinput, "loadfact\n") == 0)
     load_fact();
   else
   if(strcmp(strinput, "xping\n")   == 0)
     strcpy(replystr, "XServ is alive\n");
   else
   if(strcmp(strinput, "displayfact\n") == 0)
     save_fact_to_tmp_file();
   else
     do_new_fact(strinput);
   xprintf("server3:Test point 1.1 [already do_new_fact]\n");
   xprintf("Rply = %s\n", replystr);
   strcpy(mesg.mesg_data, replystr);
```

```
    n = strlen(mesg.mesg_data);
    mesg.mesg_data[n] = '\0';
    mesg.mesg_len = n;
    mesg.mesg_type = (long)clientpid;
    mesg_send(id, &mesg);    /* send reply to client */

    mesg.mesg_type = 1L;
    if( (n = mesg_recv(id, &mesg)) <= 0)
        xprintf("Server : Input read error\n");
    mesg.mesg_data[n] = '\0';
    xprintf("server3:mes.data=%s", mesg.mesg_data);
    xprintf("server3:Test point 1.2\n");
    get_client_data(mesg.mesg_data, &clientpid, strinput);
  }
  strcpy(mesg.mesg_data, "Server stopped\n");
  n = strlen(mesg.mesg_data);
  mesg.mesg_data[n] = '\0';
  mesg.mesg_len = n;
  mesg.mesg_type = (long)clientpid;
  mesg_send(id, &mesg);    /* send reply to client */
  system("sleep 2");
  xprintf("server3:Test point 3\n");
  if(msgctl(id, IPC_RMID, (struct msqid_ds *) 0) < 0)
    xprintf("Can't RMID message queue \n");
}
send2client(int id, long clientpid, char *str)
{  int  n;

  strcpy(mesg.mesg_data, str);
  n = strlen(mesg.mesg_data);
```

```
    mesg.mesg_data[n] = '\0';
    mesg.mesg_len = n;
    mesg.mesg_type = (long)clientpid;
    mesg_send(id, &mesg);    /* send reply to client */
}
get_client_data(char *message, int *clientpid, char *strinput)
{  int  i = 0;
   int  j = 0;
   char  strtmp[1024];

   while(message[i] != '|')
   {  strtmp[i] = message[i];
      i++;
   }
   xprintf("server3 : Test 2 message = %s\n", message);
   strtmp[i] = '\0';
   *clientpid = atoi(strtmp);
   xprintf("server3 : Test 2.1 clientpid = %d\n", *clientpid);
   i++;  /* skip '|' */
   while(message[i] != '\0')
   {  strinput[j] = message[i];
      i++;
      j++;
   }
   strinput[j] = '\0';
   xprintf("server3 : Test 2.2 strinput = %s\n", strinput);
}
```

```
/* ---------------------------------------------------------------
  Program   Sorting
  File name  sort.c
  By        M_Sukit
  Project    the.prj
  Created   94-07-02
  Modify
  Call
--------------------------------------------------------------- */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "define.h"

/* #define MAX_TABENTRY 100 */

/* ---------------------------------------------------------------
  Function : sort_cvlfile -> sort cvl file by
                                    1. load .cvl file into memory
                                    2. call sort_cvltab to sort it
                                    3. rewrite cvltab in to .cvl file
  By      : M_Sukit
  File name : sort.c
  Created  : 94-07-23
  Modify   :
  Call     : sort_cvltab
--------------------------------------------------------------- */
```

```
sort_cvlfile(char *fname)
{ FILE *fptr;
  char var[VAR_LENGTH][MAX_TABENTRY];
  char value[VAR_LENGTH][MAX_TABENTRY];
  char str[256];
  long        size;
  int         i = 0;

  if((fptr = fopen(fname, "r")) == NULL)
  {   xprintf("[SIN] : Cannot open file %s for read data\n",fname);
        return(-1);
  }
  fseek(fptr, 0, SEEK_SET);
  while(fgets(str, 255, fptr) != NULL)    /*  while not EOF */
  {
    strcpy(var[i], str);
    if(fgets(str, 255, fptr) != NULL)   /* get value from cvl file */
    {  strcpy(value[i], str);
         i++;
    }
    else
    {  xprintf("Data err in cvl file\n");
         return(-1);
    }
  }
  strcpy(var[i], "");
  sort_cvltab(var, value);
  fclose(fptr);
  i = 0;
```

```
  if((fptr = fopen(fname, "w")) == NULL)
  {   xprintf("[SIN] : Cannot open file %s for write data\n",fname);
        return(-1);
  }
  while(strcmp(var[i],"") != 0)
  {  fprintf(fptr,"%s%s",var[i], value[i]);
     i++;
  }
  fclose(fptr);
}

/* ---------------------------------------------------------
  Function  : sort_cvltab -> sort cvl file that already loaded
                          in to memory
  By        : M_Sukit
  File name : sort.c
  Created   : 94-07-23
  Modify    :
  Call      :
--------------------------------------------------------- */
sort_cvltab(char vartab[VAR_LENGTH][MAX_TABENTRY], char
valuetab[VAR_LENGTH][MAX_TABENTRY])
{ char vartemp[VAR_LENGTH];    /* use for tempory vartab before swap */
  char valuetemp[VAR_LENGTH];
  int       i;
  int       ret;
  int       swapping = 1;

  while(swapping == 1)    /* while still has swapping */
  { swapping = 0;
```

```
    i = 0;
    while( (strcmp(vartab[i+1], "") != 0) && (i < MAX_TABENTRY) )
    {   ret = strcmp(vartab[i], vartab[i+1]);
          if(ret > 0)          /*  if vartab[i] > vartab[i+1]  swap them */
          {   strcpy(vartemp, vartab[i]);
              strcpy(vartab[i], vartab[i+1]);
              strcpy(vartab[i+1], vartemp);

              strcpy(valuetemp, valuetab[i]);
              strcpy(valuetab[i], valuetab[i+1]);
              strcpy(valuetab[i+1], valuetemp);

              swapping = 1;
          }
          i++;
    }
  }
}

/* ---------------------------------------------------------------
  Function  : sort_vartab
  By        : M. Sukit
  File name : sort.c
  Created   : 94-07-23
  Modify    :
  Call      :
--------------------------------------------------------------- */

sort_vartab(char *vartab[MAX_TABENTRY], char *valuetab[])
```

```
{  char vartemp[9];    /*  use for tempory vartab before swap  */
   char valuetemp[9];
   int         i;
   int         ret;
   int         swapping = 1;

   while(swapping == 1)    /*  while still has swapping  */
   { swapping = 0;
     i = 0;
     while( (strcmp(vartab[i+1], "") != 0) && (i < MAX_TABENTRY) )
     {   ret = strcmp(vartab[i], vartab[i+1]);
         if(ret > 0)         /*  if vartab[i] > vartab[i+1]  swap them */
         {   strcpy(vartemp, vartab[i]);
             strcpy(vartab[i], vartab[i+1]);
             strcpy(vartab[i+1], vartemp);

             strcpy(valuetemp, valuetab[i]);
             strcpy(valuetab[i], valuetab[i+1]);
             strcpy(valuetab[i+1], valuetemp);

             swapping = 1;
         }
         i++;
     }
   }
}
```

```
/* ---------------------------------------------------------------
  Program     Syntax Analyzer
              Check structure of rule
  File name   syntax.c
  By          M_Sukit
  Project     the.prj
  Created     94-05-21
  Modify      95-02-12  change char to char
  Call        err() in error.c
--------------------------------------------------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "lex.h"

extern long rule_position;    /*  position of new rule in rbase  */
extern FILE *fptr;

syntax_analyzer(char *istr)
{  FILE        *cvlfptr;     /* variable cvl file      */
   char tempstr[256];  /* temp string            */
   char offset[30];    /* offset in byte point to rule that has */
                               /* specific variable name  */
   char cvl_fname[10]; /* cvl file name (a-z)    */
   int i = 0;          /* counter for istr       */
   int j = 0;          /* counter for tempstr    */
   int if_flag = 0;    /* if find 'if' in rule then if_flag++    */
   int then_flag = 0;  /* if find 'then' in rule then then_flag++ */
   int ret = 0;        /* keep the return value   */
```

```
while(istr[i] != '\0')
{ switch (istr[i])
  {  case (IF) :  if(i != 0)       /* if must place at 1st char */
                  {  err("[SYN] : if much be located at the first character in rule\n");
                     if_flag++;
                     i++;
                     ret -= 1;
                  }
                  else
                  {  if_flag++;
                     i++;
                  }
                  break;
     case (THEN): if(if_flag == 0)  /* no "if" before "then"     */
                  {  err("[SYN] : No if before then\n");
                     then_flag++;
                     i++;
          ret -= 1;
                  }
                  else
                  {  i++;
                     then_flag++;
                  }
                  break;
     case (ELSE): if(then_flag == 0)      /* no "then" before "else" */
                  {  err("[SYN] : No then before else\n");
                     then_flag++;
                     i++;
          ret -= 1;
```

```
                    }
                    else
                    {  i++;
                       then_flag++;
                    }
                    break;
        case (CALL): if(then_flag != 1) /* then should exist before CALL */
                    {  err("[SYN] : 'then' must exist before 'call'\n");
                       ret -= 1;
                    }
                    i++;    /*  skip 'CALL' */
                    if(istr[i] == '(')
                    {  while( (istr[i] != ')') && (istr[i] != '\0') )
                       i++;
                       if(istr[i] != ')')
                       {  err("[SYN] : Missing ')' in then statement\n");
                          return(-1);
                       }
                    }
                    else
                    {   err("[SYN] : After 'call' should be the '('\n");
                        return(-1);
                    }
                    break;
          default : i++;
          break;
   }
}
if (!if_flag)
{  err("[SYN] : 'if' is not exist in this rule\n"); /*  Have no 'if'  */
```

```
        ret -= 1;
    }
    if (!then_flag)
    {   err("[SYN] : 'then' is not exist in this rule\n");  /* Have no 'then' */
        ret -= 1;
    }
    return(ret);
}
```

```
/* ------------------------------------------------------------
  Program      Syntax Analyzer (from Syntax Analyzer 4)
                 Syntax Analyzer 4 (use recursion)
                                  change 'if' to while in front of
                                  islogic in check_semantic();
  File name    syntax2.c
  By           M_Sukit
  Created      94-06-05
  Modify       94-06-08
                 solve ->  (X=5&(Y=4))
                           (X=5&Y=4)
                           ((X=5)&Y=4)
                           ((X=5)(Y=4))
                           ((X=1)&(Y=2)
                 94-06-11
                 solve ->  ((X=1)&(Y=2)&(A=3))

                 cannot solve if(())then()
     change char to char
  Call         util.c

------------------------------------------------------------ */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include "lex.h"
```

```
int check_semantic(char *str, int *i, char phase)
{  char VAR[9];
   char OPER;
   char VALUE[10];
   int  result = 0;
   int  temp;

   if((str[*i] == CALL) && (phase == THEN)) /* call can exist only in then statement */
   {   (*i)++;                    /*  skip 'CALL' */
       if(str[*i] == '(')
       {   while( (str[*i] != ')') && (str[*i] != '\0') )
                 (*i)++;
           if(str[*i] != ')')
           {  err("[CSM] : Missing ')' in then statement\n");
              return(-1);
           }
       }
       else
       {   err("[CSM] : After 'call' should be the '('\n");
           return(-1);
       }
   }
   if(isvar(str, *i))
   {  getvar(VAR, str, i);
      if(isoper(str, *i))
      {  OPER = str[*i];
         (*i)++;
         if(isvalue(str, *i))
         {  getvalue(VALUE, str, i);
```

```
            if(calculate(VALUE , OPER, "0") == -1)   /* check oper */
              result += -1;
            while(islogic(str, *i))
            { OPER = str[*i];
              (*i)++;          /* Skip operator */
              if(calculate("0", OPER, "0") == -1)   /* check oper */
                result += -1;
              result += check_semantic(str, i, phase);
            }
            return(result);
          }
          else
          { err("[CSM] : After operator should be a value\n");
            return(-1);
          }
        }
        else
        { err("[CSM] : After variable should be operator\n");
          return(-1);
        }
}
else
if(str[*i] == '(')
{ (*i)++;
        result += check_semantic(str, i, phase);
        if(str[*i] != ')')
        { err("[CSM] : Missing ')'\n");
          return(-1);
        }
        else (*i)++;       /* Skip ')' */
```

```
        while(islogic(str, *i))
        {  OPER = str[*i];
           (*i)++;          /* Skip operator */
           if(calculate("0", OPER, "0") == -1)   /* check oper */
             result += -1;
           result += check_semantic(str, i, phase);
        }
    }
    return(result);
}

int semantic_analyzer(char *str)
{   int  i=1;
    int  ret = 0;

    ret = check_semantic(str, &i, IF);     /* check if statement   */
    i++;                                   /* skip 'then'          */
    ret += check_semantic(str, &i, THEN);  /* check then statement */
    return(ret);
}
```

```
/* ---------------------------------------------------------------
  Program    Do then statement
  File name  then.c
  By         M_Sukit
  Project    the.prj
  Created    94-07-26
  Modify
  Call
--------------------------------------------------------------- */
#include <stdio.h>
#include <stdlib.h>    /* for system() */
#include <string.h>
#include "lex.h"
#include "define.h"

extern FILE *explanptr;
extern FILE *eventptr;
extern char *vartab[MAX_TABENTRY];
extern char *varq[MAX_Q_DEPT];
extern char *valuetab[MAX_TABENTRY];
extern char *valueq[MAX_Q_DEPT];
extern char replystr[4080];
extern int  next_avail;
extern int  qtail;

int do_then(char *str, int *i)
{  char VAR[9];
   char VALUE[10];
   char syscall[256];  /* arg of 'CALL' */
   char tempstr[256];  /* use for lex2text for debug */
```

```
int  value, var_value;
int  var_pointer;          /* pointer to value VAR in vartab */
int  sci;                  /* pointer for syscall string     */

if(str[*i] == CALL)
{   (*i)++;         /*  skip 'CALL' */
      if(str[*i] == '(')
      {   sci = 0;
          (*i)++;    /*  skip '('    */
          while( (str[*i] != ')') && (str[*i] != '\0') )
          {   syscall[sci] = str[*i];
                sci++;
                (*i)++;
          }
          if(str[*i] != ')')
          {   err("[DOT] : Missing ')' in then statement\n");
                return(-1);
          }
          syscall[sci] = '\0';
          lex2text(str, tempstr);              /* for debug */
          tempstr[0] = ' ';      /* kill rule status , for degug */
          fprintf(eventptr,"#Rule   : %s\n", tempstr);   /* for event file */
    fflush(eventptr);
          fprintf(explanptr,"#Rule   : %s\n", tempstr);  /* for explan file */
    fflush(explanptr);
          system(syscall);
      }
      else
      {   err("[DOT] : After 'call' should be the '('\n");
          return(-1);
```

```
        }
}
if(isvar(str, *i))
{   getvar(VAR, str, i);
        strcat(VAR,"\n");
    xprintf("Test then1 : VAR=%s", VAR);
        if(str[*i] == EQ)     /*  if equal sign */
        {   (*i)++;           /*  skip '='     */
       xprintf("Test then2 : VAR=%s", VAR);
            if(isvalue(str, *i))
            {   lex2text(str, tempstr);              /* for debug */
           xprintf("Test then4 : VAR=%s", VAR);
                    tempstr[0] = ' ';      /* kill rule status , for degug */
           xprintf("Test then4.2 : VAR=%s", VAR);
                    fprintf(explanptr,"#Rule   : %s\n", tempstr);       /* for explan file */
    fflush(explanptr);
           xprintf("Test then4.3 : VAR=%s", VAR);
                    fprintf(eventptr,"#Rule   : %s\n", tempstr);       /* for event file */
    fflush(eventptr);
           xprintf("Test then5 : VAR=%s", VAR);
                    getvalue(VALUE, str, i);
                    strcat(VALUE, "\n");
                    var_pointer = search_var_table(VAR);
           xprintf("Test then6 : VAR=%s", VAR);
                    if(var_pointer == -1)       /*  if VAR is exist in vartab */
                    {   insert_newvar(VAR);     /*  add a new var (VAR)       */
                       var_pointer = search_var_table(VAR);
                    }
                    strcpy(valuetab[var_pointer], VALUE);  /* then update its value */
                    strcpy(valueq[qtail], VALUE);
```

```
            strcpy(varq[qtail], VAR);
            qtail++;
        xprintf("Test then6 : VAR=%s", VAR);
            if(qtail == MAX_Q_DEPT)  /*  circular queue  */
              qtail = 0;
            VALUE[strlen(VALUE)-1] = '\0';
            VAR[strlen(VAR)-1]    = '\0';
            fprintf(eventptr, " Result : %s = %s\n", VAR, VALUE);/* for event */
  fflush(eventptr);
            fprintf(explanptr, " Result : %s = %s\n", VAR, VALUE);/* for explan */
  fflush(explanptr);
  sprintf(replystr, "%s%s=%s\n", replystr, VAR, VALUE);   /* reply string*/
            printf("then: Result : %s = %s\n", VAR, VALUE);/* for debug */

            while(islogic(str, *i))
            { if(str[*i] == AND)
              {  (*i)++;          /* Skip operator */
                 do_then(str, i);
              }
              else
              err("[DOT] : In then statement cannot has %c sign\n",str[*i]);
            }
            return(0);    /*  function working complete */
         }
         else xprintf("[DOT] : After operator should be numberic value\n");
       xprintf("Test then3 : VAR=%s", VAR);
       }
       else xprintf("[DOT] : In then statement, after variable should be '='\n");
}
else
```

```
    if(str[*i] == '(')
    {   (*i)++;
        do_then(str, i);
        if(str[*i] != ')')
        {   xprintf("[DOT] : Missing ')'\n");
            return(-1);
        }
        else (*i)++;     /* Skip ')' */
        while(islogic(str, *i))
        {  if(str[*i] == AND)
           {   (*i)++;          /* Skip operator */
               do_then(str, i);
           }
           else
           err("[DOT] : In then statement cannot has %c sign\n",str[*i]);
        }
    }
    return(0);     /*  function working complete */
}
```

```
/* ---------------------------------------------------------------
  Program     Utilities modules
  File name   util.c
  By          M_Sukit
  Project     the.prj
  Created     94-05-29
  Modify
  Call
  Contain     get_firstvar()
              isvar()
              getvar()
              isoper()
              isvalue()
              getvalue()
              islogic()
              calculate()
              lex2text()
              rule_has_used()
              append_cvlfile()
--------------------------------------------------------------- */

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "lex.h"
#include "define.h"

extern long rule_used_list[MAX_USED_LIST];
extern int last_used;
extern long rule_position;
```

```
extern FILE *fptr;


/*------ get_firstvar() ------------*/
/*          get first variable from str */
/*      begin at ith position       */
/*------------------------------------*/


get_firstvar(char *varname, char *str, int i)
{   int j = 0;


    while((str[i] !=  '\0') && (!isupper(str[i])))
     i++;
    if(str[i] == '\0')
    {  xprintf("No variable found\n");
          return(1);
    }
    else
    {  while((isupper(str[i])) && (str[i] != '\0'))
          {   varname[j] = str[i];
            i++;
            j++;
          }
          varname[j] = '\0';
    }
}


int isvar(char *str, int i)
{  if(isupper(str[i]))
     return(1);
   else
```

```
    return(0);
}
int getvar(char *VAR, char *str, int *i)
{  int j = 0;
   int varlength = 0;

   while(isupper(str[*i]))
   {  if(varlength < 8)
         {  VAR[j] = str[*i];
           j++;
           varlength++;
         }
         (*i)++;
   }
   VAR[j] = '\0';
}
int isoper(char *str, int i)
{  char oper[10] = {"&|!=:>#<$"};
  int j;
  int ret = 0;

  for(j = 0;j < MaxOPER; j++)
    if(str[i] == oper[j])
       ret=1;
  return(ret);

}
int isvalue(char *str, int i)
{  if(  (str[i] != ')') && (str[i] != '(') && (str[i] != '\0')
    && (str[i] != AND) && (str[i] != OR)  && (str[i] != EQ)
```

```
    && (str[i] != NEQ) && (str[i] != GT)  && (str[i] != GTE)
    && (str[i] != LT ) && (str[i] != LTE))
          return(1);
  else
          return(0);

}

int getvalue(char *value, char *str, int *i)
{  int j = 0;
   int valuelength = 0;

     while(isvalue(str, *i))
     {   if(valuelength < 8)
           {   value[j] = str[*i];
             j++;
             valuelength++;
           }
           (*i)++;
     }
     value[j] = '\0';
}
int islogic(char *str, int i)
{   if( (str[i] == AND) || (str[i] == OR) )
      return(1);
    else
      return(0);
}
int calculate(char *VALUE1, char oper, char *VALUE2)
{
```

```
switch(oper)
{ case AND:  return((int)VALUE1 && (int)VALUE2);
  case OR :  return((int)VALUE1 || (int)VALUE2);
  case NOT:
                  break;
  case EQ :  if(strcmp(VALUE1, VALUE2) == 0)
                    return(1);
                  else
                    return(0);
  case NEQ:  if(strcmp(VALUE1, VALUE2) != 0)
                    return(1);
                  else
                    return(0);
  case GT :  if(strcmp(VALUE1, VALUE2) > 0)
                    return(1);
                  else
                    return(0);
  case GTE:  if(strcmp(VALUE1, VALUE2) >= 0)
                    return(1);
                  else
                    return(0);
  case LT :  if(strcmp(VALUE1, VALUE2) < 0)
                    return(1);
                  else
                    return(0);
  case LTE:  if(strcmp(VALUE1, VALUE2) <= 0)
                    return(1);
                  else
                    return(0);
  default :  err("[CAL] : Unknow operator %c\n",oper);
```

```
                return(-1);
    }
}

lex2text(char *lex, char *text)
{  int i = 0;
   char tempstr[5];

   strcpy(text,"");        /* initialize output string */
   while(lex[i] != '\n')
   {   switch(lex[i])
          {  case IF    :  strcat(text,"if ");
                           break;
             case THEN  :  strcat(text," then ");
                           break;
             case ELSE  :  strcat(text," else ");
                           break;
             case CALL  :  strcat(text," call ");
                           break;
             case AND   :  strcat(text," && ");
                           break;
             case OR    :  strcat(text," || ");
                           break;
             case EQ    :  strcat(text,"=");
                           break;
             case NEQ   :  strcat(text," != ");
                           break;
             case GT    :  strcat(text," > ");
                           break;
             case GTE   :  strcat(text," >= ");
```

```
                    break;
            case LT   : strcat(text," < ");
                    break;
            case LTE  : strcat(text," <= ");
                    break;
            default   : sprintf(tempstr,"%c",lex[i]);
                    strcat(text, tempstr);
                    break;
        }
        i++;
    }
}
rule_has_used(char *var_pos)
{   int i = 0;

    while((i < 256) && (i <= last_used))
    {   if(rule_used_list[i] == atol(var_pos))
            return(1);      /* this rule is used before */
        else
            i++;
    }
    return(0);      /*  not found this rule in rule list */
}

addto_cvlfile(char *istr)
{  FILE       *cvlfptr;
   char       tempstr[256];
   char       cvl_fname[12];
   int        i = 0;      /*  pointer for input string  */
   int        j;          /*  pointer for temp string   */
```

```
int         then_flag = 0;
int         var_flag = 1 ; /* indicate that the following is var */

xprintf("util.addto_cvlfile : begin\n");
while((then_flag == 0) && (istr[i] != '\0'))
{  while((!isupper(istr[i])) && (istr[i] != '\0'))
   {  /*i++;          go to next char until not upper case */
            if(istr[i] == THEN)
              then_flag = 1;
     xprintf("%c\n", istr[i]);
     if(isoper(istr,i) || islogic(istr,i))
     {  xprintf("util.addto_cvlfile : \n");
       var_flag ^= 1;
     }
     i++;
   }
   if((istr[i] != '\0') && (then_flag == 0) && (var_flag == 1))
   /* if not EOL ,not then and is variable*/
   {  j = 0;
            while( isupper(istr[i]) )
            {   tempstr[j] = istr[i];
              i++;
              j++;
            }
            tempstr[j] = '\n';
          tempstr[j+1] = '\0';
          xprintf("Variable = %s",tempstr);
          sprintf(cvl_fname,"%c.cvl",tempstr[0]);
          if((cvlfptr = fopen(cvl_fname,"a")) == NULL)
          {   xprintf("Cannot open file %s\n", cvl_fname);
```

```
            xprintf("You cannot add any more rule\n");
            return(-1);
          }
          else
          {  fprintf(cvlfptr,"%s%ld\n",tempstr,rule_position);
            insert_newvar(tempstr);
            fclose(cvlfptr);
          }
      }
      else i++;
  }
  return(0);
}

append_cvlfile(char *istr)
{ FILE       *cvlfptr;
  char       tempstr[256];
  char       cvl_fname[12];
  int        i = 0;      /*  pointer for input string  */
  int        j;          /*  pointer for temp string   */
  int        then_flag = 0;
  int        var_flag = 1; /* indicate this position is var */

  while((then_flag == 0) && (istr[i] != '\0'))
  {  xprintf("util.append1 : %c, O%d, L%d\n", istr[i], isoper(istr, i), islogic(istr, i));
     while((!isupper(istr[i])) && (istr[i] != '\0'))
     { /*i++;          go to next char until not upper case */
          if(istr[i] == THEN)
              then_flag = 1;
       if(isoper(istr,i) || islogic(istr,i))
```

```
  { xprintf("util.append : \n");
    var_flag ^= 1;
  }
xprintf("util.append2 : %c, O%d, L%d\n", istr[i], isoper(istr, i), islogic(istr, i));
  i++;
}
if((istr[i] != '\0') && (then_flag == 0) && (var_flag == 1))
/* if not EOL and not then */
{ j = 0;
      while( isupper(istr[i]) )
        { tempstr[j] = istr[i];
          i++;
          j++;
        }
        tempstr[j] = '\n';
        tempstr[j+1] = '\0';
        xprintf("Variable = %s",tempstr);
        sprintf(cvl_fname,"%c.cvl",tempstr[0]);
    xprintf("Test util1 : fname=%s\n", cvl_fname);
    xprintf("Test util2 : rule_pos=%ld\n", rule_position);
        if((cvlfptr = fopen(cvl_fname,"a")) == NULL)
        { xprintf("Cannot open file %s\n", cvl_fname);
          xprintf("You cannot add any more rule\n");
          return(-1);
        }
        else
        { fprintf(cvlfptr,"%s%ld\n",tempstr,rule_position);
          insert_newvar(tempstr);
          fclose(cvlfptr);
        }
```

```
      }
      else i++;
    }
    return(0);
}
```

```
/* ------------------------------------------------------------
  Program     Expert Server [xSrv]
  File name   xsrv.c
  By          M_Sukit
  Project     xsrv.prj
  Created     94-11-19
  Modify
------------------------------------------------------------ */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "define.h"

FILE *fptr;             /*  Rule base file   */
FILE *explanptr;        /*  Explanation file */
FILE *eventptr;         /*  Event file       */
long rule_position;     /*  position of rule in rbase  */
long rule_used_list[MAX_USED_LIST]; /*  contain list of rules that already used in
                                       inference engine */
char *vartab[MAX_TABENTRY];
char *varq[MAX_Q_DEPT];  /* stack for keep var name for infer */
char *valuetab[MAX_TABENTRY];
char *valueq[MAX_Q_DEPT];       /* stack for keep value of varq  */
char replystr[4080];      /*  Reply string for send output to client    */
int  next_avail = 0;      /*  Next available position in vartab         */
int  qtail = 0;           /*  tail pointer for fw_chainning queue       */
int  qhead = 0;           /*  head pointer for fw_chainning queue       */
int  last_used = 0;       /*  pointer of rule_use_list, specify the last
                                rule used */
int  lineno = 0;
```

```
main()
{  char str[256];
   int qid = 0;

   initialize();
   xprintf(":");
   do_exp_server();
}
initialize()
{ FILE *cvlptr;
  char str[256];
  char oldstr[256] = "none";
  char name;            /*  File name not include . (a - z) */
  char fullname[8];     /*  File name include .             */
  int  i;

  xprintf("Initialize...\n");
  /*-------- ALLOCATE VARTAB ----------*/
  for(i=0; i<MAX_TABENTRY ; i++)
     vartab[i] = malloc(10);
  i = 0;
  /*------- INITIALIZE VARTAB ---------*/
  for(i=0; i<MAX_TABENTRY ; i++)
     strcpy(vartab[i], "zzzzzzzz");
  i = 0;
  /*------- ALLOCATE VALUETAB ---------*/
  for(i=0; i<MAX_TABENTRY ; i++)
     valuetab[i] = malloc(10);
  i = 0;
```

```
/*------ INITIALIZE VALUETAB --------*/
for(i=0; i<MAX_TABENTRY ; i++)
   strcpy(valuetab[i], "null");
i = 0;
/*-------- ALLOCATE VARQUEUE---------*/
for(i=0; i<MAX_Q_DEPT ; i++)
   varq[i] = malloc(9);
i = 0;
/*------- ALLOCATE VALUEQUEUE--------*/
for(i=0; i<MAX_Q_DEPT ; i++)
   valueq[i] = malloc(9);
i = 0;
/*--- INITIALIZE RULE USED LIST -----*/
for(i=0; i<MAX_USED_LIST ; i++)
   rule_used_list[i] = -1;
i = 0;
/*------ OPEN RBASE FILE --------*/
if((fptr = fopen("rbase.xsrv", "r")) == NULL)
{  fptr = fopen("rbase.xsrv","w");   /* if rbase file not exist then */
   fclose(fptr);                    /* create it                    */
   if((fptr = fopen("rbase.xsrv","r")) == NULL)
   {  xprintf("Cannot open rbase.xsrv file\n");
         exit(1);
   }
}
fseek(fptr,0,SEEK_END);
rule_position = ftell(fptr);
/*------ OPEN EXPLANATION FILE --------*/
init_explan_file();
/*------ OPEN EVNET LOG FILE ----------*/
```

```
if((eventptr = fopen("event.log", "a")) == NULL)
{  xprintf("Cannot open event log file, program terminate\n");
   exit(-1);
}
/*------- OPEN CVL FILE ---------*/
name = 'A';
while((int)name <= (int)'Z')              /* repeat for file name A-Z .cvl */
{ sprintf(fullname,"%c.cvl",name);
  if((cvlptr = fopen(fullname, "r+")) != NULL)   /* if ?.cvl is exist */
  {  xprintf("Processing [%s] \n",fullname);
        sort_cvlfile(fullname);
        while(fgets(str, 255, cvlptr) != NULL)
        { if(strcmp(oldstr, str) != 0)
          { strcpy(vartab[i], str);
            next_avail++;
            i++;
          }
          strcpy(oldstr, str);
          fgets(str, 255, cvlptr);     /* Skip position of this var */
        }
  }
  fclose(cvlptr);
  name++;
 }
 xprintf("Initialize complete\n");
}
```

ภาคผนวก ค. ตัวอย่างโปรแกรม XClient

```
/* ---------------------------------------------------------------
  Program  : client3.c
  By       : M_Sukit
  Project  : iesh.prj
  Created  : 94-12-05
  Modify   :
  Call     :
  Project  : None
  For      : Read data from file and send to xSrv
--------------------------------------------------------------- */
#include <stdio.h>
#include <string.h>
#include "srv.h"

Mesg mesg;

main()
{  int  id;

   if( (id = msgget(MKEY1, 0)) < 0)
      printf("client: can't msgget message queue 1\n");
   client(id);
}

client(id)
int id;
{   FILE *datafptr;
    int  n;
    int  pid;   /* process id of this process */
```

```
    char strtmp[1024];   /* tempory string */

    if((datafptr = fopen("datafile", "r")) == NULL)
    {   printf("client3 : Cannot open datafile\n");
        exit(-1);
    }
    pid = getpid();
    while(fgets (strtmp, MAXMESGDATA, datafptr) != NULL)
    {   sprintf(mesg.mesg_data, "%d|%s", pid, strtmp);
        n = strlen(mesg.mesg_data);
        mesg.mesg_data[n] = '\0';
        printf("Input is %s\n", mesg.mesg_data);
        mesg.mesg_len = n;
        mesg.mesg_type = 1L;
        mesg_send(id, &mesg);
        if(strcmp(strtmp, "exit\n") == 0)
        {   printf("CLIENT : Program terminated [normal]\n");
            exit(1);
        }
        mesg.mesg_type = (long)pid;
        if( (n = mesg_recv(id, &mesg)) >= 0)
          printf("%d, %s\n", n, mesg.mesg_data);
        else
          printf("data read error\n");
    }
}
```

ประวัติผู้เขียน

นายสุกิจ เมฆจำเริญ เกิดเมื่อวันที่ 22 กุมภาพันธ์ 2512 ที่จังหวัดกรุงเทพฯ สำเร็จการศึกษาวิศกรรมศาสตร์บัณฑิต (วิศวกรรมคอมพิวเตอร์) จากสถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง ปีการศึกษา 2534 และเข้าทำงานที่การไฟฟ้าฝ่ายผลิตแห่งประเทศไทย ตำแหน่งวิศวกรระดับ 4 ปัจจุบันดำรงตำแหน่งผู้จัดการแผนก System Engineer ฝ่ายบริการและสนับสนุน บริษัท โซลูชั่นส์ คอร์ปอเรชั่น จำกัด