ต้นแบบการประชุมอิเลคทรอนิกส์

โดย

นาย กฤษฎา ทองอร่าม
นาย เชษฐพล รัตนานนท์
นาย ธวัชชัย ศิริวัฒโนทัย

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตร์บัณฑิต
สาขาวิชาคณิตศาสตร์ประยุกต์
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2538

Prototype of Electronics Conference

Presented by

Mr. Krissada Thongaram

Mr. Chettapol Rattananonta

Mr. Tawatchai Siriwatnotai

A Special Project Submitted in Partial Fulfilment of the

Requirement for the Degree of Bachelor of Science

Department of Applied Mathematics and Computer Sciece

Faculty of Science

King Mongkut 's Institute of Technology Ladkrabang

1995

หัวข้อโครงงานพิเศษ โปรแกรมต้นแบบการประชุมอิเลคทรอนิคส์

โดย 1. นายกฤษฎา ทองอร่าม รหัส 35504102

2. นายเชษฐพล รัตนานนท์ รหัส 35504106

3. นายธวัชชัย ศิริวัฒโนทัย รหัส 35504107

ภาควิชา คณิตศาสตร์และวิทยาการคอมพิวเตอร์

อาจารย์ที่ปรึกษา อาจารย์ศรัณย์ อินทโกสุม

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้โครงงานนี้เป็นส่วนของการศึกษาตามหลักสูตรวิทยาศาสตร์บัณฑิต

_____

(รองศาสตราจารย์ภัคคินี ชิตสกุล) หัวหน้าภาควิชา

คณะกรรมการพิจารณาโครงงานพิเศษ

_____

(อาจารย์สิริลักษณ์ เตียพิริยะกิจ) ประธานกรรมการ

_____

(รองศาสตราจารย์ภัคคินี ชิตสกุล) กรรมการ

_____

(อาจารย์ศรัณย์ อินทโกสุม) กรรมการและอาจารย์ที่ปรึกษา

บทคัดย่อ

ปัญหาพิเศษ นี้มีวัตถุประสงค์เพื่อพัฒนาโปรแกรมต้นแบบการประชุมอิเล็คทรอนิกส์ขึ้น ซึ่งโปรแกรมต้นแบบนี้พัฒนาขึ้นโดยใช้ภาษาซีบนระบบปฏิบัติการยูนิกซ์ โดยลักษณะของการประชุมจะมีการกำหนดเลขานุการของการประชุม ซึ่งเลขานุการจะทำการระบุผู้ร่วมเข้าประชุมในกลุ่ม และกำหนด ประธานของการประชุม

ผลจากปัญหาพิเศษจะได้โปรแกรมที่มีความสามารถในการทำงานคล้ายกับการประชุมจริงบนระบบเครือข่าย

Abstract

The purpose of this project is to develop Prototype of Electronics Conference software. Is developed by C language under UNIX operating system. The conference must has one secretary. Then secretary of each group will list the person who allowed to join conference and define president of each group. As the result, it can simulate a real conference through computer network.

กิตติกรรมประกาศ

ปัญหาพิเศษฉบับนี้สำเร็จลงได้ด้วยดี เพราะได้รับความกรุณาโดยเฉพาะอย่างยิ่งจากบุคคลต่าง ๆ เหล่านี้

1) อาจารย์ศรัณย์ อินทโกสุม ท่านช่วยแนะนำแก้ไขปัญหา และช่วยให้แนวคิดในการเขียนโปรแกรมรวมทั้งการจัดหาเอกสารต่าง ๆ ที่เกี่ยวข้องกับการทำปัญหาพิเศษ
2) อาจารย์ปัญญาพล หอระดะ ท่านช่วยให้คำแนะนำและแนะแนวทางในการเขียนโปรแกรมและแก้ไขปัญหาที่ เกิดขึ้นระหว่างการทำปัญหาพิเศษ
3) อาจารย์ทุก ๆ ท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้ และ ให้คำแนะนำมาโดยตลอด 4 ปี
4) บิดา มารดา และผู้มีอุปการคุณ ที่ท่านให้โอกาสและส่งเสริม ทางด้านการศึกษา
5) เพื่อนๆ และ น้องๆ คณะวิทยาศาสตร์ ที่ ให้คำแนะนำ และเป็นแรงใจมาตลอด
6) เจ้าหน้าที่ภาควิชาคณิตศาสตร์ ฯ ทุกท่านที่สนับสนุนในการใช้ห้องปฏิบัติการคอมพิวเตอร์ และให้ความสะดวกในการเบิกอุปกรณ์ต่าง ๆ ที่ใช้ในการวิจัย

ขอขอบคุณในความกรุณาอย่างสูง

คณะผู้จัดทำ

สารบัญรูปภาพ

สารบัญตาราง

สารบัญ

บทที่ 1

บทนำ

1.1) ความเป็นมาของปัญหา

ในปัจจุบันนี้แม้ว่าการเดินทางไปที่ไหน ๆ จะสะดวกสบายมากกว่าสมัยก่อนก็ตาม แต่ถ้าในมุมมองของธุรกิจแล้วจะมองว่า ในการเดินทางไปติดต่อกับผู้อื่นมักจะเสียเวลาในการเดินทาง โดยเวลาที่เสียไปนั้นในบางครั้งอาจจะไม่คุ้มกับจำนวนเงินหรือประโยชน์ ที่ ควรจะได้รับ และยิ่งในเมืองใหญ่ ๆ หลายเมืองมักจะมีปัญหาในการเดินทางสูง เช่นรถติดซึ่งจะทำให้เสียเวลามากขึ้น การเดินทางแต่ละครั้งอาจจะทำให้สุขภาพจิดแย่ลงไปกว่าเดิมอีกด้วย ซึ่งจะส่งผลทำให้การทำงานไม่มีประสิทธิภาพเท่าที่ควร การนำเอาการประชุมอิเล็คทรอนิกส์มาใช้ในชีวิตประจำวัน จะช่วยขจัดปัญหาเหล่านี้ได้โดยในบางครั้งอาจไม่จำเป็นที่จะต้องเดินทาง จากบ้านไปเข้าประชุม โดยอาจจะทำเพียงแค่เปิดเครื่องคอมพิวเตอร์ในบ้านแล้วต่อเข้าไปยังที่ประชุม (ในกรณีนี้เครื่องคอมพิวเตอร์ของคุณจะต้องต่อเข้ากับเครือข่ายคอมพิวเตอร์ที่สามารถติดต่อไปยังที่ประชุมนั้น ๆ ได้ โดยที่ประชุมนั้น ๆ อาจจะไม่จำเป็นต้องเป็นที่ทำงาน)ก็สามารถที่จะเข้าร่วมประชุมได้

โดยการประชุมอิเล็คทรอนิกส์ (Electronics Conference) คือการประชุมหรือการติดต่อทางไกลระหว่างคนหลาย ๆ คนในเวลาเดียวกันโดยผ่านทางอุปกรณ์อิเล็คทรอนิกส์เช่น คอมพิวเตอร์ โดยมีขั้นตอนการทำงานเหมือนกับการประชุมกันตรง ๆ ทุกอย่าง เช่น การบันทึกผลการประชุม การควบคุมองค์ประชุม ตลอดจนการรักษาความปลอดภัยของผลการประชุมไม่ให้รั่วไหลออกไปได้ ฯลฯ

1.2) วัตถุประสงค์ของการวิจัย

1. เพื่อศึกษาการเขียนโปรแกรมที่ใช้ติดต่อกันในระบบเครือข่ายประชุม
2. เพื่อจัดทำโปรแกรมสำหรับต้นแบบการประชุมอิเล็กทรอนิกส์

1.3) ขอบเขตของการวิจัย

1. สร้างต้นแบบการประชุมอิเล็กทรอนิกส์ กรณีองค์ประชุมมากที่สุด 12 คน
2. มีผู้ควบคุมการประชุม(Administrator)ในการระบุ ชื่อกลุ่ม ชื่อเลขาฯ และ รหัสคีย์ของการประชุม
3. ใช้ภาษาอังกฤษในการประชุม

1.4) ขั้นตอนการวิจัย

1. ศึกษาการทำงานของระบบปฏิบัติการยูนิกซ์
2. ศึกษาการเขียนโปรแกรมภาษาซีบนระบบปฏิบัติการยูนิกซ์

3. ศึกษาการเขียนโปรแกรมเพื่อติดต่อกันบนระบบปฏิบัติการยูนิกซ์
4. จัดทำต้นแบบ

1.5) ประโยชน์ที่คาดว่าจะได้รับ

1. ได้โปรแกรมต้นแบบของการประชุมอิเล็คทรอนิกส์
2. เป็นแนวทางในการศึกษา และ พัฒนาระบบการประชุมอิเล็คทรอนิกส์ให้มีประสิทธิภาพต่อไป

บทที่ 2

ความหมายและทฤษฎีที่ใช้

ระบบปฏิบัติการยูนิกซ์(UNIX)

ยูนิกซ์เป็นระบบปฏิบัติการแบบหลายผู้ใช้ (Multiuser) หลายงาน (Multitask) สำหรับการใช้งานทั่วไป คอมพิวเตอร์ที่ใช้ยูนิกซ์สามารถรองรับโปรแกรมได้มากกว่าหนึ่งโปร -แกรมในเวลาเดียวกัน เช่นคุณอาจกำลังใช้แอพพลิเคชั่นฐานข้อมูล ในขณะเดียวกับที่ผู้ร่วมงานกำลังใช้เวิร์ดโปรเซสเซอร์บนคอมพิวเตอร์เครื่องเดียวกันทั้งคู่จะใช้สถานีปลายทาง(Terminal) ในการเข้าถึงยูนิกซ์คอมพิวเตอร์ที่เป็นแม่ข่าย

หัวใจของยูนิกซ์คือเคอร์เนล (kernel) ของระบบปฏิบัติการแบบจัดสรรเวลา (Time-sharing operating system) ซอฟต์แวร์ระบบปฏิบัติการจะควบคุมทรัพยากรของคอมพิวเตอร์และจัดสรรให้แก่แอพพลิเคชั่นที่ทำงานอยู่บนคอมพิวเตอร์นั้น โปรแกรมเชลล์ (shell) จะทำการโต้ตอบกับผู้ใช้ เพื่อให้คุณสามารถใช้โปรแกรม คัดลอกไฟล์ ลงบันทึกเข้าและออก และทำงานอื่น ๆ เชลล์อาจแสดงในรูปบรรทัดคำสั่งอย่างง่าย หรือการติดต่อกับผู้ใช้แบบกราฟิกด้วย ไอคอนและหน้าต่าง ไม่ว่าในกรณีใดเชลล์และแอพพลิเคชั่นที่ทำงานบนยูนิกซ์ จะใช้บริการของเคอร์เนลเพื่อจัดการไฟล์และอุปกรณ์รอบข้าง

โปรเซสในระบบยูนิกซ์

โปรเซส (Process) คือโปรแกรมที่กำลังทำงานหรือถูกจัดการ (execute, เอกซิคิวต์) ซึ่งเป็นปัจจัยสำคัญในเครือข่ายคอมพิวเตอร์ โดยระบบปฏิบัติการของเครื่องคอมพิวเตอร์ เมื่อเราพูดถึงเครื่องคอมพิวเตอร์ 2 เครื่องกำลังติดต่อซึ่งกันและกันเราจะหมายความถึงโปรเซสสองโปรเซสกำลังติดต่อซึ่งกันและกัน

ในระบบปฏิบัติการบางตัวใช้คำว่าทาสก์ (task) แทนโปรเซส ระบบปฏิบัติการหนึ่ง ๆ ที่มีการงานแบบมัลติทาสกิ้ง (multitasking) สามารถที่จะเอกซิคิวต์โปรเซสได้มากกว่า 1โปร -เซส ในเวลาเดียวกัน และยังสามารถทำให้โปรแกรมหนึ่ง ๆ ถูกเอกซิคิวต์ได้หลาย ๆ โปรเซสในเวลาเดียวกันด้วย เช่น อาจจะมีผู้ใช้ 10 คนกำลังใช้งานโปรแกรม ๆ เดียวกัน

ส่วนประกอบของโปรเซส

- หมายเลขประจำโปรเซส (PROCESS ID , PID)

ทุกๆ โปรเซสจะต้องมีหมายเลขประจำโปรเซสที่ไม่ซ้ำกับของโปรเซสอื่น ๆ โดยหมายเลขนี้เป็นเลขจำนวนเต็ม มีค่าตั้งแต่ 0 - 30000 ซึ่งเคอร์เนลจะกำหนดค่า PID เมื่อโปรเซสถูกสร้างขึ้น โปรเซสที่มีค่า PID เป็น 1 คือโปรเซสที่เรียกว่าโปรเซส init โปรเซสที่มีค่า PID เป็น 0 คือเคอร์เนลโปรเซส หรืออาจเรียกว่า swapper หรือ scheduler โปรเซสที่เป็นเครื่องมือที่ใช้จัดการเกี่ยวกับระบบหน่วยความจำเสมือน (virtual memory) ของยูนิกซ์จะมีค่า PID เป็น 2 ซึ่งเป็นเคอร์เนลโปรเซสที่เรียกว่า Pagedaemon ส่วนโปรเซสที่มีหมายเลขนอกเหนือจากนี้จะไม่มีความหมายใดเป็นพิเศษ

- หมายเลขประจำโปรเซสแม่ (PARENT PROCESS ID , PPID)

ทุก ๆ โปรเซสจะต้องมีโปรเซสแม่ (parent process) และจะต้องมีค่าหมายเลขประจำโปรเซสแม่ เคอร์เนลจะเป็นตัวกำหนดค่าหมายเลขโปรเซสแม่เมื่อโปรเซสใหม่ถูกสร้างขึ้น

- หมายเลขประจำตัวผู้ใช้ (REAL USER ID)

ผู้ใช้แต่ละคนจะถูกกำหนดหมายเลขประจำตัวผู้ใช้เป็นเลขจำนวนเต็มบวก หมายเลขประจำตัวผู้ใช้จะถูกนำมาใช้ในกรณีตัวอย่างเช่น ในไฟล์ระบบที่จะบันทึกค่าเจ้าของไฟล์ ซึ่งนอกจากชื่อล็อคอิน บนระบบยูนิกซ์แล้ว ยังมีการกำหนดหมายเลขประจำตัวผู้ใช้ที่ไม่ซ้ำกันให้กับผู้ใช้แต่ละคนด้วย

- หมายเลขประจำกลุ่มผู้ใช้ (REAL GROUP ID)

เช่นเดียวกับค่าหมายเลขประจำตัวผู้ใช้ ผู้ใช้ทุกคนจะต้องถูกกำหนดค่าหมายเลขประจำกลุ่มผู้ใช้ โดยทั่วไปหมายเลขประจำกลุ่มผู้ใช้จะใช้สำหรับรวมผู้ใช้ระบบยูนิกซ์ เป็นกลุ่มหรือแผนกซึ่งมีข้อแตกต่างจากหมายเลขประจำตัวผู้ใช้คือ สามารถมีผู้ใช้ได้หลายคนที่มีค่าหมายเลขประจำกลุ่มผู้ใช้เหมือนกัน ในไฟล์ /etc / group จะเก็บความสัมพันธ์ระหว่างชื่อกลุ่มกับหมายเลขประจำกลุ่มผู้ใช้ ในแต่ละไฟล์บนระบบจะต้องบันทึกค่าหมายเลขประจำกลุ่มผู้ใช้ของเจ้าของไฟล์ โดยทั้งค่าหมายเลขประจำตัวผู้ใช้และหมายเลขประจำกลุ่มผู้ใช้ของไฟล์จะถูกใช้ในระบบยูนิกซ์สำหรับที่จะอนุญาตให้เข้าไปใช้ไฟล์

- หมายเลขประจำกลุ่มโปรเซส (PROCESS GROUP ID)

ทุก ๆ โปรเซสเป็นสมาชิกของกลุ่มของโปรเซสที่กำหนดโดยค่าตัวเลขจำนวนเต็มบวกของหมายเลขประจำกลุ่มโปรเซสซึ่งมีข้อแตกต่างไปจากหมายเลขประจำโปรเซสคือ หมายเลขประจำโปรเซสจะมีค่าเฉพาะของแต่ละโปรเซสซึ่งจะไม่ซ้ำกัน ส่วนหมายเลข

ประจำกลุ่มโปรเซสอาจจะเหมือนกันในหลาย ๆ โปรเซสได้ โดยโปรเซสที่มีหมายเลขประจำกลุ่มโปรเซสเหมือนกันนี้จะเรียกว่าโปรเซสเหล่านั้นเป็นสมาชิกของกลุ่มของโปรเซสเดียวกัน

ถ้าหมายเลขประจำกลุ่มโปรเซสไปเหมือนกับค่าหมายเลขประจำโปรเซสของโปรเซสใดในกลุ่ม จะเรียกโปรเซสนั้น ๆ ว่าโปรเซสผู้นำกลุ่ม(process group leader) แต่ถ้าค่าหมายเลขทั้งสองไม่เหมือนกันในแต่ละโปรเซสแล้วหมายเลขประจำกลุ่มโปรเซสของทุก ๆ โปรเซสจะเป็นหมายเลขประจำโปรเซสของโปรเซสผู้นำกลุ่ม

หมายเลขประจำกลุ่มสถานีปลายทาง และการควบคุมสถานีปลายทาง (TERMINAL GROUP ID AND CONTROL TERMINAL)

ทุก ๆ โปรเซสจะต้องเป็นสมาชิกของกลุ่มสถานีปลายทางซึ่งแทนค่าโดยตัวเลขจำนวนเต็มบวกของหมายเลขประจำกลุ่มสถานีปลายทาง ซึ่งตัวเลขดังกล่าวคือค่าหมายเลขประจำโปรเซสของโปรเซสผู้นำกลุ่มที่ทำการเปิดเครื่องปลายทางนั้น ๆ เหมือนกับ login shell ที่ถูกเรียกเมื่อต้องการที่จะล็อคอิน (login) เข้าไปในระบบยูนิกซ์ โปรเซส นี้จะถูกเรียกว่าโปรเซสควบคุม (control process) สำหรับสถานีปลายทาง และสถานีปลายทางหนึ่ง ๆ จะมีโปรเซสควบคุมได้เพียงโปรเซสเดียว

ค่าหมายเลขประจำกลุ่มสถานีปลายทางกำหนดลักษณะของการควบคุมสถานีปลายทางของกลุ่มของโปรเซสซึ่งการควบคุมสถานีปลายทางนี้ใช้สำหรับส่งสัญญาณเมื่อค่าจากสถานีปลายทางถูกส่งผ่านและเมื่อล็อคอินเชลล์ถูกยกเลิก เมื่อโปรเซสซึ่งเป็นทั้งโปรเซสผู้นำกลุ่มและโปรเซสควบคุม สำหรับสถานีปลายทางใด ๆ เรียกใช้ exit สำหรับยกเลิกการทำงาน สัญญาณ hangup จะถูกส่งไปยังทุก ๆ โปรเซสในกลุ่มของโปรเซสนั้น ๆ ทำให้การควบคุมสถานีปลายทางถูกอ้างถึงโดยอัตโนมัติโดยอุปกรณ์ที่มีชื่อว่า /dev / tty ถ้าโปรแกรมต้องการจะเก็บไว้ก็ทำได้โดยอ่านหรือเขียนลงไปยังการควบคุมสถานีปลายทางเพื่อที่จะติดต่อกับอินพุต , เอาต์พุต มาตรฐานอีกครั้ง

สัญญาณ (SIGNALS)

สัญญาณคือการส่งข้อความถึงโปรเซสเกี่ยวกับเหตุการณ์ที่เกิดขึ้นในบางครั้งเรียกว่าซอฟต์แวร์อินเทอร์รัพต์ (software interrupts) โดยปรกติจะเกิดขึ้นโดยไม่มีจังหวะซึ่งหมายความว่าโปรเซสจะไม่รู้ว่าจะเกิดสัญญาณในเวลาใด โดยสัญญาณนี้สามารถส่งได้ 2 แหล่งคือ

- จากโปรเซสหนึ่งไปยังโปรเซสอื่น ๆ หรือตัวเอง
- จากเคอร์เนลไปยังโปรเซส

คำสั่งที่ใช้ในการจัดการโปรเซส

คำสั่ง FORK

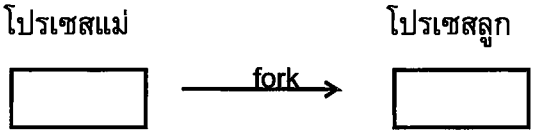
ในยูนิกซ์มีทางเดียวที่จะสร้างโปรเซสใหม่ขึ้นมาเพื่อที่จะนำมาเอกซิคิวต์คือใช้ fork system call

```
int  fork();
```

โดย fork จะถูกนำมาเอกซิคิวต์เพื่อสร้างชุดก๊อปปี้ของโปรเซส โดยโปรเซสที่เป็นตัวเอกซิคิวต์คำสั่ง fork เรียกว่าโปรเซสแม่ (parent process) และโปรเซสที่เกิดขึ้นใหม่เรียกว่าโปรเซสลูก (child process) คำสั่ง fork ถูกเรียกจากโปรเซสหนึ่ง (โดยโปรเซสแม่) แต่จะส่งค่าคืนมาให้สองโปรเซส (โปรเซสแม่ และโปรเซสลูก) ค่าที่ส่งกลับโดยการเรียกใช้คำสั่ง fork มีค่าได้แตกต่างกันออกไป คือ ส่งค่ากลับเมื่ออยู่ในสถานะที่เป็นโปรเซสแม่จะส่งค่าเป็นค่า PIDของ โปรเซสที่เกิดขึ้นมาใหม่ หรือเมื่ออยู่ในสถานะที่เป็นโปรเซสลูกจะส่งค่ากลับมาเป็น 0 หรือถ้าทำคำสั่งนี้ไม่สำเร็จ (ไม่มีการสร้างโปรเซส) ก็จะส่งค่ากลับมาเป็น -1 ตัวอย่างของ fork system call แสดงได้ดังนี้

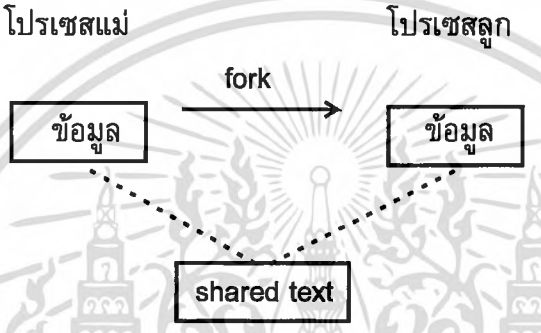
```
main()
{
        int   childpid;
        if ( (childpid = fork()) == -1) {
                perror (" can't fork");
                exit(1);
        }
        else if (childpid == 0) {
                printf("child  : child pid = %d , parent pid = %d \n", getpid() ,
                getppid());
                exit (0);
        }
        else {
                printf("parent :  child  pid  = %d , parent  pid = %d \n" ,childpid
                                                                , getpid( ) );
                exit(0);
        }
}
```

เราแสดงการทำงานของ fork ได้คือ



รูปที่ 2.1 แสดงการทำงานของโปรเซส

ถ้าเมื่อส่วน text ของโปรเซสสามารถใช้ร่วมกันได้แล้ว ทั้งโปรเซสแม่และโปรเซสลูกก็สามารถที่จะใช้ส่วน text ร่วมกันหลังจากเกิดการ fork



รูปที่ 2.2 แสดงการใช้ส่วน text ร่วมกันของโปรเซสแม่และ โปรเซสลูก

และจากความจริงที่ว่าเมื่อเกิดการ fork ส่วนข้อมูลของโปรเซสลูกเหมือนกับส่วนข้อมูลของโปรเซสแม่ ดังนั้นจึงไม่จำเป็นที่จะต้องก๊อปปี้จากไฟล์ของโปรแกรมที่อยู่ในดิสก์

ลักษณะที่สำคัญอย่างหนึ่งของ fork คือไฟล์ที่ถูกเปิดขึ้นจากโปรเซสแม่ก่อนที่จะมีการ fork จะถูกใช้ร่วมกันกับโปรเซสลูกหลังจากการ fork ซึ่งลักษณะนี้ทำให้ง่ายสำหรับการที่โปรเซสแม่เปิดไฟล์หรืออุปกรณ์แล้วส่งผ่านมายังโปรเซสลูก หลังจากเกิด fork แล้วโปรเซสแม่ก็จะทำการปิดไฟล์ของตัวเองที่เปิดขึ้นมาเพื่อโปรเซสลูก ดังนั้นจึงทำให้ทั้งสองโปรเซสไม่ได้ใช้ไฟล์ร่วมกัน

จากค่าตัวแปรของโปรเซสที่กำหนดไว้ในดอนต้นสามารถแบ่งเป็นตัวแปรที่มีค่าเหมือนกันทั้งในโปรเซสลูกและโปรเซสแม่คือ

- หมายเลขประจำตัวผู้ใช้
- หมายเลขประจำกลุ่มผู้ใช้
- หมายเลขประจำกลุ่มโปรเซส
- หมายเลขประจำกลุ่มสถานีปลายทาง
- ไดเร็กทอรี่ราก
- ไดเร็กทอรีทำงานปัจจุบัน

และตัวแปรที่มีค่าไม่เหมือนกันระหว่างโปรเซสแม่และโปรเซสลูกคือ

- โปรเซสลูกมีค่าหมายเลขประจำโปรเซสใหม่และไม่ซ้ำกัน
- โปรเซสลูกมีค่าหมายเลขประจำโปรเซสแม่ที่แตกต่างออกไป
- โปรเซสลูกมีชุดก๊อปปี้ของ ไฟล์เดสคริปเตอร์ (file descriptor) ของตัวเอง
- อายุตลอดจนเวลาตามสัญญาณนาฬิกาถูกตั้งค่าเป็น 0 ในโปรเซสลูก

สาเหตุที่ทำให้เกิด fork มี 2 ข้อคือ

1. โปรเซสหนึ่ง ๆ ต้องการที่จะจำลองตัวเองเพื่อให้มีชุดหนึ่งสามารถติดต่อกับ fork หรือควบคุมการกระทำของ fork และชุดที่เหลือจึงจะไปทำทาสก์อื่นซึ่งเป็นลักษณะของเซิร์ฟเวอร์ของเครือข่ายทั่วๆ ไป
2. โปรเซสต้องการที่จะเอกซิคิวต์โปรแกรมอื่น ๆ เพราะว่าทางที่จะสร้างโปรเซสใหม่มีทางเดียวคือโดยใช้ fork โปรเซสจะต้อง fork ครั้งแรกเพื่อสร้างชุดคัดลอกของตัวเอง แล้วโปรเซสที่เกิดขึ้นใหม่นี้จะเรียกใช้คำสั่ง exec สำหรับเอกซิคิวต์โปรแกรมใหม่ซึ่งจะมีลักษณะเหมือนกับเป็นเปลือกหุ้มโปรแกรมอีกชั้นหนึ่ง

คำสั่ง EXIT

การจะหยุดการทำงานของโปรเซสโดยเรียกใช้ exit system call ซึ่ง system call นี้จะไม่ส่งค่ากลับ เมื่อ exit ถูกเรียกค่าตัวเลขที่จะแสดงสถานะ exit จะถูกผ่านจากโปรเซสไปยังเคอร์เนลโดยค่าสถานะ exit นี้สามารถส่งผ่านไปให้กับโปรเซสแม่ของโปรเซสที่หยุดการทำงาน เมื่อ โปรเซสแม่นั้น ๆ เรียกใช้ wait system call โดยที่ค่า 8 บิตล่างของตัวเลขสถานะ exit จะถูกใช้ นั่นคือโปรเซสจะหยุดการทำงานด้วยค่าสถานะ exit ที่มีขอบเขตตั้งแต่ 0 - 255 โดยกำหนดว่าโปรเซสที่หยุดการทำงานอย่างปรกติจะมีค่า exit เป็น 0 แต่ถ้าค่าที่ได้ไม่ใช่ 0 แสดงว่าเกิดกรณีผิดพลาด

คำสั่ง EXEC

ยูนิกซ์สามารถเอกซิคิวต์โปรแกรมได้เพียงทางเดียวคือการใช้ exec system call ผ่านทางโปรเซสที่กำลังทำงานอยู่ โดย exec จะแทนที่โปรเซสนั้น ๆ ด้วยโปรแกรมใหม่โดยที่หมายเลขโปรเซสไม่เปลี่ยนแปลง เราเรียกโปรเซสที่ใช้คำสั่ง exec ว่า calling process และโปรแกรมที่ถูก exec ว่า new program

ฟังก์ชั่น exec มีลักษณะที่แตกต่างกัน 6 รูปแบบคือ

```
int execlp(char *filename , char * arg0, char *arg1,....., char *argn , (char *) 0);

int execl ( char *pathname , char *arg0 , char *arg1,....., char *argn , (char *) 0);
```

int execle (char *pathname , char *arg0, char *arg1,..... , char * argn, (char *) 0 ,

char **envp);

int execvp (char *filename , char ** argv);

int execv (char *pathname , char ** argv);

int execve (char *pathname , char ** argv , char **envp);

ฟังก์ชั่น exec ส่งค่ากลับมาเมื่อเกิดข้อผิดพลาด แต่ถ้าไม่มีข้อผิดพลาดก็จะทำการควบคุม new program

การมองฟังก์ชั่นทั้ง 6 ฟังก์ชั่น

1. 3 ฟังก์ชั่นแถวบนมีการกำหนดแต่ละอาร์กิวเมนต์ ในลักษณะที่มีการแยกทีละอาร์กิว - เมนต์สำหรับฟังก์ชั่น และใช้ NULL พอยน์เตอร์สำหรับยกเลิกการกำหนดตัวแปรที่ระบุจำนวนของอาร์กิวเมนต์ ส่วน 3 ฟังก์ชั่นในแถวที่ 2 สร้างอาร์กิวเมนต์ในลักษณะที่เป็นอะเรย์สำหรับพอยน์เตอร์ชี้ไปยังอาร์กิวเมนต์ โดย argv อะเรย์นี้จะเก็บค่า NULL เมื่อข้อมูลสิ้นสุดซึ่งไม่จำเป็นต้องนับจำนวนข้อมูล
2. 2 ฟังก์ชั่นในคอลัมน์ทางซ้ายกำหนดอาร์กิวเมนต์ในรูป filename ซึ่งแปลงให้อยู่ในรูปของ pathname โดยมีการใช้ตัวแปร PATH สำหรับสภาพแวดล้อมปัจจุบัน ถ้าไม่มีตัวแปร PATH แล้วค่ามาตรฐานจะถูกกำหนดให้เป็น " :/bin: / user/bin " ถ้าอาร์กิวเมนต์ pathname ใน execlp หรือ execvp มีเครื่องหมาย / ภายในชุดตัวอักษรแล้ว ตัวแปร PATH จะไม่ถูกนำมาใช้ ส่วน 4 ฟังก์ชั่นทางขวามือมีการกำหนดในลักษณะที่เป็น pathname เต็มรูปแบบ
3. 4 ฟังก์ชั่นในคอลัมน์ทางซ้ายมือทั้ง 2 คอลัมน์ไม่ได้กำหนดพอยน์เตอร์สภาพแวดล้อมไว้แน่นอน แต่จะใช้ค่าปัจจุบันของตัวแปรสภาพแวดล้อมภายนอกในการที่จะสร้างสภาวะของ new program ส่วน 2 ฟังก์ชั่นทางขวามือได้มีการกำหนดสภาพแวดล้อมไว้แน่-นอน ซึ่งค่า envp จะถูกยกเลิกการใช้เมื่อกำหนดค่าเป็น NULL

โปรแกรมที่ถูก exec จะรับค่าตัวแปรเหล่านี้จากโปรเซสคือ

- หมายเลขประจำโปรเซส
- หมายเลขประจำโปรเซสแม่
- หมายเลขประจำกลุ่มโปรเซส
- หมายเลขประจำกลุ่มสถานีปลายทาง
- ช่วงเวลาตั้งแต่เริ่มต้นจนถึงสัญญาณนาฬิกาปัจจุบัน
- ไดเร็กทอรี่ราก
- ไดเร็กทอรีทำงานปัจจุบัน
- หมายเลขประจำตัวผู้ใช้

- หมายเลขประจำกลุ่มผู้ใช้
- ล็อคไฟล์

คำสั่ง WAIT

โปรเซสหนึ่ง ๆ สามารถรอการทำงานของโปรเซสลูกจนกว่าจะเสร็จสิ้นโดยใช้ wait system call

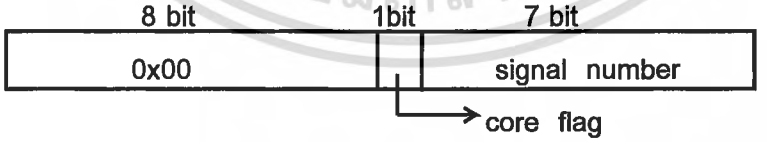
int wait (int *status)

wait จะส่งค่าหมายเลขโปรเซสของโปรเซสลูกที่ยกเลิก ถ้าโปรเซสที่เรียกใช้ wait ไม่มี โปร - เซส ลูกก็จะส่งค่ากลับมาเป็น -1 ถ้าโปรเซสที่เรียกใช้ wait มีโปรเซสลูกโปรเซสหนึ่งหรือหลาย โปรเซสที่ยังไม่หยุดการทำงานแล้ว โปรเซสนั้น ๆ จะถูกพักการทำงานโดยเคอร์เนลจนกว่าโปร- เซส ลูกของมันจะหยุดการทำงาน เมื่อโปรเซสลูกหยุดการทำงานและคำสั่ง wait ส่งค่ากลับ ถ้าค่าสถานะไม่เป็นค่า NULL ค่าที่ส่งผ่านไปยัง exit ซึ่งเกิดจากการหยุดการทำงานของโปรเซส ลูกจะถูกเก็บลงในตัวแปรสถานะ นอกจากนี้ข้อมูลก็จะส่งค่ากลับโดยคำสั่ง wait ด้วย มีเหตุ - การณ์ที่เกิดขึ้นได้ 3 เหตุการณ์สำหรับคำสั่ง wait ในการที่จะส่งค่าหมายเลขโปรเซสเป็นค่าส่ง กลับ

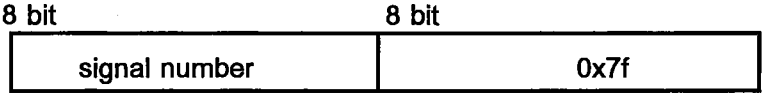
1. โปรเซสลูกเรียกใช้คำสั่ง exit
2. โปรเซสลูกถูกยกเลิกโดยสัญญาณ (signal)
3. โปรเซสลูกถูกเขียนทับและโปรเซสหยุดการทำงาน ซึ่งเหตุนี้เกิดขึ้นเมื่อมีโปรเซส หนึ่ง ๆ มีการทับกันในการเอกซิคิวต์กับโปรเซสอื่น ๆ เช่นเมื่อใช้ debugger ในการ เช็คการทำงานที่ละขั้นตอนของโปรเซส

ในแต่ละกรณีข้างต้น จะมีข้อมูลในตัวแปรสถานะแตกต่างกันออกไป ดังรูปที่ 2.3

โปรเซสลูกเรียกใช้ exit



โปรเซสถูกยกเลิกโดยสัญญาณ



โปรเซสหยุดการทำงาน

รูปที่ 2.3 แสดงสถานะของโปรเซส

หมายเลขสัญญาณ (signal number) จะมีค่ามากกว่า 0 มิฉะนั้นจะไม่สามารถที่จะทำให้แตกต่างกันทั้ง 3 กรณี บิต core flag จะมีค่าเป็น 1 ถ้า การยกเลิกการทำงานของโปรเซสทำให้เกิด core file ส่วนในกรณีอื่น จะมีค่าเป็น 0

จากที่กล่าวมาข้างต้นโปรเซสจะมีโปรเซสแม่เพียงโปรเซสเดียวแต่จะมีโปรเซสลูกได้หลายๆ โปรเซส เคอร์เนลจะทำให้เกิดกรณีต่อไปนี้เมื่อโปรเซสหนึ่ง ๆ exit

- ถ้าโปรเซสแม่ของโปรเซสที่กำลัง exit ได้มีการเรียกใช้คำสั่ง wait โปรเซสแม่ก็จะได้รับการแจ้งว่า โปรเซสลูกได้ยกเลิกการทำงานแล้ว (อาจจะเกิดจากการที่คำสั่ง wait ส่งค่ากลับมายังโปรเซสแม่) โปรเซสแม่จะได้รับสถานะ exit ของโปรเซสลูกถ้า อาร์กิว - เมนต์ status ของโปรเซสแม่ไม่มีค่าเป็น 0
- ถ้าโปรเซสแม่ของโปรเซสที่กำลัง exit ไม่มีการเรียกใช้คำสั่ง wait โปรเซสที่จะยกเลิกการทำงานดังกล่าวจะถูกกำหนดให้เป็นซอมบี้โปรเซส (zombie process) ซึ่งเป็นโปร-เซสที่ยกเลิกตัวเองแต่โปรเซสแม่ไม่ได้มีการรอการยกเลิกนี้ เคอร์เนลจะลบทรัพยากร ทุกอย่างของโปรเซสนี้ทั้งหมด เช่นหน่วยความจำ แต่จะยังคงสถานะ exit ของโปรเซสจนกว่าโปรเซสแม่จะเรียกใช้ wait
- ค่าหมายเลขโปรเซสแม่ของโปรเซสลูกที่ยกเลิกตัวเองจะตั้งค่าเป็น 1 (เป็นโปรเซส init)
- ถ้าหมายเลขประจำโปรเซส , หมายเลขประจำกลุ่มโปรเซสและหมายเลขประจำกลุ่มสถานีปลายทางของโปรเซสที่ยกเลิกมีค่าเท่ากัน สัญญาณ hangup (SIGNUP) จะถูกส่งให้กับแต่ละโปรเซสที่มีค่าหมายเลขประจำกลุ่มโปรเซสเท่ากับของโปรเซสที่ยกเลิก

จะเกิดอะไรขึ้นกับค่าหมายเลขโปรเซสแม่ของโปรเซสลูกถ้าโปรเซสแม่ยกเลิกการทำงานก่อนโปรเซสลูก ซึ่งเราจะมาพิจารณาถึงเหตุการณ์ใน 2 กรณีดังนี้คือ

1. ถ้าโปรเซสลูกยกเลิกการทำงานก่อนโปรเซสแม่จะเป็นกรณีที่ปรกติเมื่อเข้าไปใช้งานเชลล์
   (a) ถ้าโปรเซสแม่มีการเรียกใช้ wait แล้ว wait จะส่งค่ากลับมายังโปรเซสแม่เป็นค่าหมายเลขโปรเซสของโปรเซสลูกที่ยกเลิก
   (b) ถ้าโปรเซสแม่ไม่ได้เรียกใช้ wait แล้วโปรเซสลูกจะถูกทำให้เป็นซอมบี้โปรเซส

2. โปรเซสแม่ยกเลิกการทำงานก่อนโปรเซสลูกจะทำให้มีปัญหาตามมา เพราะว่าหมายเลขโปรเซสแม่ ของโปรเซสลูกจะมีค่าผิดพลาดเมื่อโปรเซสแม่ยกเลิกการทำงาน ในกรณีนี้ยูนิกซ์จะทำการตรวจสอบโปรเซสทุกโปรเซสว่าค่าของหมายเลข โปรเซสแม่ของโปรเซสใดมีค่าเหมือนกับหมายเลขโปรเซสของโปรเซสที่จะถูกยกเลิก ซึ่งโปร - เซสเหล่านั้นก็จะกลายเป็นโปรเซสกำพร้า ยูนิกซ์จะตั้งค่าหมายเลข โปรเซสแม่

ของโปรเซสเหล่านั้นให้เป็น 1 ซึ่งเป็นหมายเลขโปรเซสของโปรเซส init ดังนั้นโปรเซส init จึงกลายมาเป็นโปรเซสแม่ของโปรเซสกำพร้าเหล่านี้ โปรเซส init ไม่มีการถูกยกเลิก ส่วนในกรณีที่หมายเลขโปรเซสแม่ถูกเปลี่ยนแปลง โปรเซสลูกที่กำลังเอกซิคิวต์ก็จะไม่ทราบว่าโปรเซสแม่ได้ถูกยกเลิกไปแล้ว

การทำงานระหว่างโปรเซส (INTERPROCESS COMMUNICATION)

ในการทำงานของระบบเครือข่ายจะต้องมีการสื่อสารกัน ระหว่างโปรเซส มากกว่าหนึ่งโปรเซส เราจึงจำเป็นที่จะต้องระมัดระวังในการใช้วิธีการต่างๆ สำหรับโปรเซสหนึ่งๆ เพื่อให้สื่อสารกับโปรเซสอื่น ๆ ได้ ซึ่งวิธีที่จะยกมาใช้คือ

MESSAGE QUEUE

ในปัจจุบันระบบปฏิบัติการสมัยใหม่จะต้องมีส่วนประกอบที่สำคัญคือการส่งข้อความระหว่างโปรเซสซึ่งส่วนใหญ่จะมีข้อกำหนดคือโปรเซสจะสามารถส่งข้อความไปยังโปรเซสที่กำหนดไว้แล้วเท่านั้น แต่ในระบบปฏิบัติการ UNIX System V สามารถลดข้อจำกัดด้านนี้ไปได้ โดยข้อความจะถูกเก็บไว้ในเคอร์เนลและข้อความแต่ละข้อความจะถูกกำหนดรหัสประจำข้อความ(Message Queue Identifies , msqid) ขึ้น ซึ่งใช้สำหรับระบุถึงลักษณะของข้อความแต่ละข้อความในการทำงาน โปรเซสจะอ่านและเขียนข้อความไปเก็บไว้ในคิว (queue) ซึ่งจะทำให้โปรเซสที่จะรับข้อความไม่ต้องรอข้อความที่จะถูกส่งมายังคิวก่อนที่โปรเซสที่ส่งจะถูกอนุญาตให้เขียนข้อความไปยังคิวนั้นได้

สรุปคือโปรเซสจะเขียนข้อความทิ้งไว้ใน Message Queue แล้วจบการทำงาน โดยไม่ต้องรอโปรเซสที่จะมาอ่านข้อความ

ส่วนประกอบของข้อความในคิว

- ชนิดมีลักษณะข้อมูลเป็นจำนวนเต็มแบบยาว (Long Integer)
- ความยาวของข้อความ (สามารถมีค่าเป็น 0 ได้)
- ข้อมูล (ถ้าข้อมูลมีความยาวมากกว่า 0)

เคอร์เนลจะดูแลข้อมูลในทุกๆ Message Queue ในระบบ ซึ่งเมื่อเขียนเป็นโครงสร้างได้ดังนี้

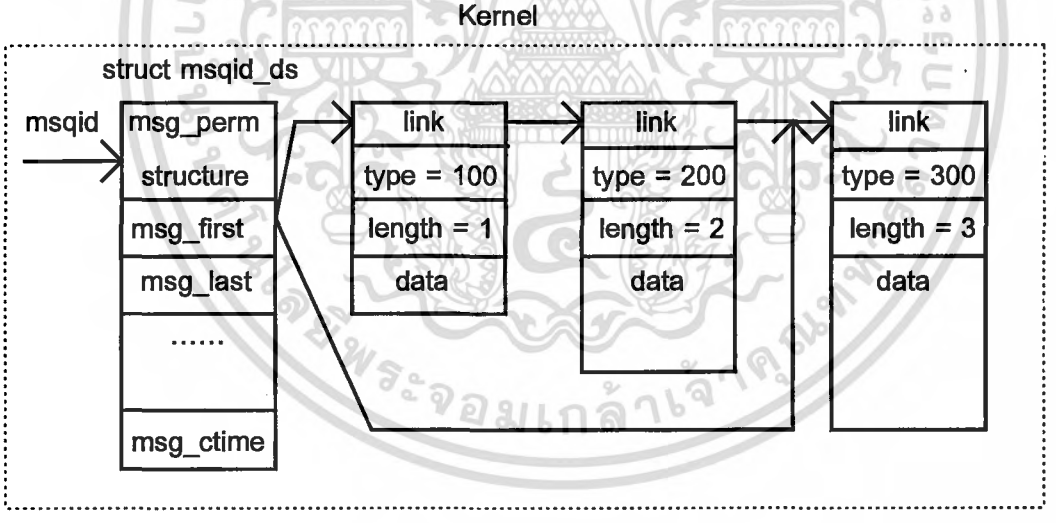
```
#include <sys/types.h>
    #include <sys/ipc.h>          /* Defines the ipc_perm structures */
    struct msqid_ds
    {
        struct ipc_perm  msg_perm;   /* operation permission struct */
        struct msg     *msg_first;   /* พอยน์เตอร์ชี้ไปยังข้อความแรกบนคิว */
```

```
        sttuct msg      *msg_last;      /* พอยน์เตอร์ชี้ไปยังข้อความสุดท้ายบนคิว */
        ushort          msg_cbytes;     /* ขนาดของข้อความปัจจุบันบนคิว */
        ushort          msg_qnum;       /* จำนวนของข้อความบนคิว */
        ushort          msg_qbytes;/*ขนาดของข้อความที่มากที่สุดที่ให้มีได้บนคิว */
        ushort          msg_lspid;      /* pid of last msgsnd */
        ushort          msg_lrpid;      /* pid of last msgrcv */
        ushort          msg_stime;      /* time of last msgsnd */
        ushort          msg_rtime;      /* time of last msgrcv */
        ushort          msg_ctime;  /* time of last msgctl  ( that change the above
) */
    };
```

ipc_perm เป็นตัวที่ตรวจสอบถึงการเข้าถึง Message Queue เฉพาะ

msg คือโครงสร้างของข้อมูลภายในซึ่งเคอร์เนลใช้สำหรับจัดการดูแลลิงค์ลิสต์ของคิว

เราสามารถแสดงลักษณะของ Message Queue ใน เคอร์เนลโดยทั่วๆไป มีลักษณะเป็นลิงค์ลิสต์ของข้อความ ดังแสดงในรูป 2.4



รูปที่ 2.4 แสดงลักษณะของ Message Queue

จากรูป 2.4 มีข้อความ 3 ข้อความอยู่ในคิวโดยมีความยาวของข้อความเป็น 1 ไบต์ , 2 ไบต์ และ3 ไบต์ และมีชนิดของข้อความเป็น 100 , 200 และ 300 ตามลำดับ

Message Queue ที่ถูกสร้างขึ้นมาใหม่หรือมีอยู่แล้วจะถูกเข้าถึงโดย msgget System call

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

```
#include <sys/msg.h>
int msgget ( key_t   key , int msgflag );
```

ค่าของ msgflag สรุปได้ดังตารางที่ 2.1

Numerics	Symbolic	Descriprtion
0400	MSG_R	Read by owner
0200	MSG_W	Write by owner
0040	MSG_R >> 3	Read by group
0004	MSG_R >> 6	Read by world
0002	MSG_W >> 6	Write by world

ตารางที่ 2.1 แสดงค่าของ msgflag

โดยปกติ msgget จะคืนค่าเป็น msqid แต่ถ้ามีข้อผิดพลาดเกิดขึ้นจะคืนค่า -1

หลังจากที่ Message Queue ถูกเปิดโดย msgget เราสามารถใส่ข้อความลงบนคิวโดยใช้ msgsnd System call

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd ( int msqid , struct msgbuf  *ptr , int length , int flag );
```

ptr เป็นตัวชี้ที่ชี้ไปยังเรคอร์ด ซึ่งมีรูปแบบดังนี้

```
struct msgbuf
{
        long    mtype ;          /* ค่าชนิดของข้อความซึ่งจะต้องมากกว่า 0 */
        char    mtext[ 1 ] ;              /* ข้อมูลในข้อความ  */
}
```

ข้อมูลที่จะใช้ในการใส่ลงคิวสามารถที่จะเป็นได้ทั้งข้อมูลแบบตัวอักษร (TEXT) และแบบไบนารี โดยที่ค่าของชนิดของข้อมูลต้องมากกว่า 0 แต่ถ้าค่าของชนิดของข้อมูลมีค่าเป็น 0 จะเป็นการระบุว่าเรียกใช้ msgrcv Stystem call จากรูปแบบ ptr ชี้ไปยังชนิดของข้อมูลซึ่งเป็นจำนวนเต็มชนิดยาวและตามด้วยข้อมูลนั้นๆ (ข้อความมีความยาวมากกว่า 0) เคอร์เนลจะไม่ตีความสารบัญของข้อความทั้งหมด แต่ถ้าตัวปฏิบัติการโปรเซสร่วม (cooperating process)

ต้องการที่จะแลกเปลี่ยนข้อความซึ่งประกอบด้วย จำนวนเต็มชนิดสั้น, ตัวอักษรขนาด 8 ไบต์ เราสามารถที่จะกำหนดโครงสร้าง ได้ดังนี้

```
typedef struct my_msgbuf
{
        long    mtype ;         /* ค่าชนิดของข้อความ  */
        short   mshort ;        /* ข้อมูลเริ่มต้นของข้อความ  */
        char    mchar[ 8 ];
} Message;
```

length ระบุถึงความยาวของข้อความ (เป็นไบต์) ซึ่งเป็นของข้อมูลที่ผู้ใช้กำหนดโดย สามารถมีค่าเป็น 0 ได้

flag จะระบุค่าเป็น IPC_NOWAIT หรือ 0 ถ้า flag มีค่าเป็น IPC_NOWAIT จะเป็นการบอกให้ system call คืนค่าไปบอกว่า ไม่มีที่ว่างใน Message Queue สำหรับข้อความใหม่ที่จะใส่เข้าไป ซึ่งกรณีนี้จะเกิดขึ้นเมื่อมีข้อความในคิวมากเกินไปหรือถ้ามีข้อความของระบบมาก ถ้าไม่มีที่ว่างพอสำหรับข้อความ IPC_NOWART จะถูกกำหนดค่า, msgsnd จะคืนค่า -1 และ errno ถูกเซตเป็น EAGAIN . แต่ถ้ามีที่ว่างสำหรับข้อความ msgsnd จะ คืนค่า 0

ข้อความที่อ่านมาจาก Message Queue จะเรียกใช้ msgrcv System call

```
#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 );
```

ptr มีลักษณะเดียวกับ msgsnd และบอกถึงตำแหน่งของข้อความที่รับมา

length บอกถึงขนาดของข้อมูลที่ถูกชี้โดย ptr ซึ่งเป็นขนาดของข้อมูลที่ใหญ่ที่สุดที่ถูกคืนค่ามาจาก system call

ถ้าบิต MSG_NOERROR ใน flag ถูกเซตค่า แสดงว่าถ้าข้อมูลของข้อความที่ได้รับมีค่ามากกว่า length จะมีการแบ่งข้อมูลออกเป็นส่วนและคืนค่าโดยปราศจากข้อผิดพลาดแต่จะไม่มีการระบุค่าของแฟล็ก MSG_NOERROR ที่ผิดพลาดเนื่องจาก length เล็กเกินไปสำหรับข้อความที่รับมาทั้งหมด

ค่าของ msgtype ที่แสดงถึงกรณีต่างๆของข้อความบนคิว

-ถ้า msgtype =0 ค่าข้อความแรกบนคิวจะถูกส่งคืนเพราะแต่ละ message queue จะทำงานแบบเข้าก่อน , ออกก่อน (first-in, first -out) ค่า msgtype เป็น 0 แสดงว่าค่าข้อความแรกสุดบนคิวได้ถูกส่งคืน

- ถ้า msgtype มากกว่า 0 ค่าข้อความแรกที่มีค่าชนิดเท่ากับค่า msgtype จะถูกส่งคืน
- ถ้า msgtype น้อยกว่า 0 ค่าข้อความแรกที่มีค่าชนิดน้อยกว่าหรือเท่ากับค่าสัมบูรณ์ของค่า msgtype จะถูกส่งคืน

flag เป็นตัวระบุที่ว่าจะทำอย่างไรชนิดของข้อความที่ถูกเรียก ไม่อยู่บนคิว ถ้า IPC_NOWAIT ถูกเซตค่า msgrcv system call จะคืนค่าทันทีถ้าข้อความไม่สามารถเรียกใช้ได้ ในกรณีนี้ system call จะคืน -1 และ errorno ถูกเซตเป็น ENOMSG ในกรณีอื่นๆดัวที่เรียกจะหยุดชั่วคราวจนกระทั่งกรณีเหล่านี้เป็นจริงคือข้อใดข้อหนึ่งคือ

- ข้อความของชนิดที่เรียกใช้งานได้
- Message Queue ถูกนำออกจากระบบ
- โปรเซสรับสัญญาณที่ถูกส่ง

msgtype	ค่าชนิดของข้อความที่ส่งคืน
0L	100
100L	100
200L	200
300L	300
-100L	100
-200L	100
-300L	100

ตารางที่ 2.2 ค่าชนิดของข้อความที่ส่งคืน

กล่าวโดยรวมคือ MSG_NOERROR ของ flag สามารถเซตค่าโดยกรณีที่กล่าวมาข้างต้น

เมื่อทำงานสำเร็จ msgrcv จะส่งค่าขนาดของข้อมูล (เป็นไบต์) ของข้อความที่ได้รับโดยยังไม่รวมกับขนาดของข้อความที่เป็นจำนวนเต็มชนิดยาวซึ่งถูกส่งค่าผ่าน ptr

msgctl system call ใช้ในการควบคุมการปฏิบัติการบน Message Queue

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl ( int msqid , int cmd , struct msqid)ds * buff );
```

แต่ส่วนใหญ่ที่ใช้จะมีเพียง cmd ของ IPC_RMID เพื่อนำ Message Queue ออกจากระบบ

การใช้หน่วยความจำร่วมกัน (SHARE MEMORY)

พิจารณาขั้นตอนโดยทั่วไปที่เกี่ยวข้องในการคัดลอก (copy) โปรแกรมของระบบไฟล์ไคลเอ็นต์-เซิร์ฟเวอร์

1.เซิร์ฟเวอร์ ทำการอ่านไฟล์ข้อมูล (input file)

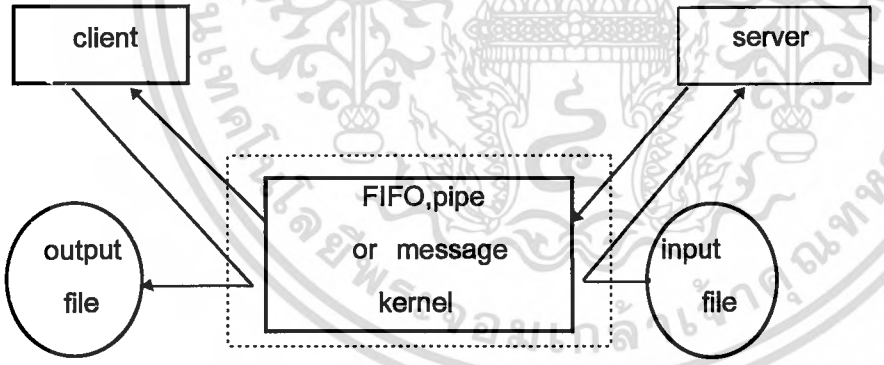
ข้อมูลต่างๆ จะอ่านโดย เคอร์เนล ในส่วนของอินเทอร์นอล บล็อค บัฟเฟอร์ (internal block buffer) และจะถูกคัดลอกมาเก็บไว้ที่บัฟเฟอร์ของเซิร์ฟเวอร์ (การอ่านโดยใช้ read - system call)

2.เซิร์ฟเวอร์จะเขียนข้อมูลลงในเมสเสจ (message)โดยมีการใช้วิธี ไฟโฟ (fifo) หรือ เมสเสจ คิว (message queue)

3.ไคลเอ็นต์จะอ่านข้อมูลจากไอพีซีเคอร์เนล (ipc kernel)โดยที่ข้อมูลจะถูกคัดลอก จาก เคอร์เนลของ ไอพีซีบัฟเฟอร์ (ipc buffer) ไปที่บัฟเฟอร์ของไคลเอ็นต์

4.สุดท้าย ข้อมูลจะถูกคัดลอก จากบัฟเฟอร์ของไคลเอ็นต์ (โดยใช้คำสั่ง write system call) เพื่อเป็นไฟล์ผลลัพธ์(output file) (โดยข้อมูลจะถูกเก็บใน เคอร์เนล บัฟเฟอร์ (kernel buffer)ก่อน พร้อมกันนั้นเคอร์เนลจะทำหน้าที่เป็นตัวเขียนที่แท้จริงลงในดีไวซ์(device)ที่เวลาต่อมา)

ทั้ง4 ขั้นตอนยังคงเป็นการทำงานระหว่าง เคอร์เนล และ ยูสเซอร์ โปรเซส (user process) การทำงานจึงใช้เวลามาก ในขณะที่การทำงานของยูนิกส์พยายามที่จะเร่งเวลาให้การคัดลอกให้เร็วมากที่สุดเท่าที่จะทำได้



รูปที่ 2.5 แสดงการเคลื่อนที่ของข้อมูลระหว่าง 2 โปรเซส

ปัญหาที่เกิดขึ้น

ปัญหาที่ 2 โปรเซส ทำการแลกเปลี่ยนข้อมูลกันโดยข้อมูลนั้นจะต้องถูกส่งไปยังเคอร์เนล

ทางแก้ไข

การใช้หน่วยความจำร่วมกันได้จัดหาวิธีการในการแก้ปัญหาโดยอนุญาติให้ 2 โปรเซสขึ้นไป สามารถร่วมกันใช้ส่วนของหน่วยความจำได้ แต่วิธีการนี้แน่นอนจะก่อให้เกิดปัญหาในกรณีที่มีหลายโปรเซส เข้ามาใช้หน่วยความจำพร้อมกัน (โปรเซส ต่างๆ จะมีระดับในการใช้หน่วยความจำของตนเท่าๆ กัน)

ทางแก้ไขคือ ถ้ามีโปรเซสหนึ่งกำลังอ่านข้อมูลบางส่วนในแชร์เมมโมรี(share memory) อยู่ โปรเซส อื่นๆต้องรอคอยการอ่านจนเสร็จสิ้นเสียก่อน เราจะแก้ปัญหานี้ โดยการใช้เซมาฟอร์(semaphore)

ดังนั้น ขั้นตอนสำหรับ ไคลเอ็นต์ -เซิร์ฟเวอร์ จึงเปลี่ยนแปลงได้ดังนี้

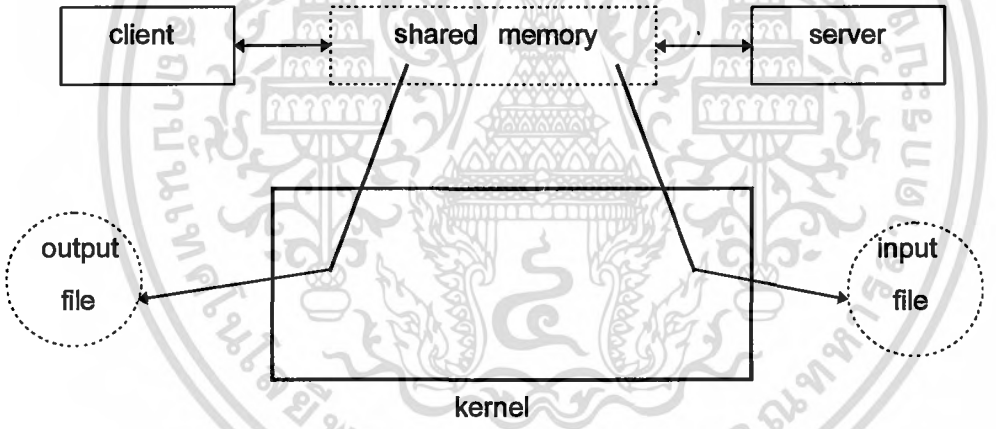
1. เซิร์ฟเวอร์ จะเข้าถึง แชร์เมมโมรี เซกเมนต์ (shared memory segment)โดยใช้เซมาฟ่อร์

2. เซิร์ฟเวอร์ อ่านข้อมูลจากไฟล์นำเข้ามาเก็บใน แชร์เมมโมรี เซกเมนต์

3. เมื่อทำการอ่านเสร็จสิ้นเซิร์ฟเวอร์แจ้งไปยังไคลเอ็นต์โดยใช้ เซมาฟอร์ อีกครั้งหนึ่ง

4.ไคลเอ็นต์จะเขียนข้อมูลจาก แชร์เมโมรี เซกเมนต์ ไปที่ไฟล์ผลลัพธ์

ดังแสดงในรูปที่ 2.6



รูปที่ 2.6 แสดงการเคลื่อนที่ของข้อมูลระหว่างไคลเอ็นต์และเซิร์ฟเวอร์โดยใช้แชร์เมโมรี

ในรูป ดังกล่าวข้อมูลถูกคัดลอก 2 ครั้งคือ

1.จาก ไฟล์นำเข้าไปที่แชร์เมโมรี

2.จาก แชร์เมมโมรีไปที่เอาท์พุท ไฟล์

ทั้งการคัดลอก 2 แบบนี้ บางที่ก็ต้องใช้บล็อคบัฟเฟอร์ของเคอร์เนล

สำหรับ ทุกๆ แชร์เมโมรี เซกเมนต์ นั้น เคอร์เนล จะเก็บไว้โดยใช้โครงสร้างข้อมูลดังต่อไปนี้

shmget system call

แชร์เมมโมรี เซกเมนต์จะถูกเพิ่มขึ้นหรือที่มีอยู่แล้วจะถูกเข้าถึงโดยใช้ shmget system call

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

โดยค่าที่คืนกลับมาของshmget คือตัวที่ชี้ไปที่แชร์เมมโมรี(shared memory identifier)ได้แก่ shmid หรือ -1 ในกรณีที่เกิดข้อผิดพลาดของขนาดของ segment ในหน่วย bytes shmflag คือการรวมกันของค่าคงที่ แสดงในตารางที่ 2.3

Numeric	Symbolic	Description
0400	SHW_R	Read by owner
0200	SHM_W	Write by owner
0040	SHM_R >>3	Read by group
0020	SHM_W>>3	Write by group
0004	SHM_R>>6	Read by world
0002	SHM_W>>6	Write by world

ตารางที่ 2.3 แสดงค่าคงที่ของ shmflag

shmat system call

shmget จะทำการการเพิ่มหรือเปิดแชร์เมมโมรี เซกเมนต์ แต่ไม่สามารถเข้าถึง เซกเมนต์ ได้ เราต้องติดต่อ แชร์เมโมรี เซกเมนต์ โดยการเรียกจาก shmat system call system call นี้จะส่งค่ากลับคือ ค่าตำแหน่งเริ่มต้นของ แชร์เมมโมรี เซกเมนต์

โดยกฎในการตัดสินใจเลือกตำแหน่ง (address)พิจารณาได้ดังนี้

1.ถ้า shmaddr มีค่าเป็น 0 ระบบเลือกตำแหน่งสำหรับผู้เรียก(caller)

2.ถ้า shmaddr ไม่เท่ากับ 0 จะส่งค่าตำแหน่งกลับโดยขึ้นอยู่กับผู้เรียกที่จะระบุค่า shm-rnd ของ shmaddr

แต่โดยการใช้งานแล้วเราจะสะดวกในการเรียก shmat ที่เจาะจงลงไปโตยshmaddr = 0 และอนุญาติให้เคอร์เนลทำการเลือกตำแหน่ง ในกรณีเกิดข้อผิดพลาดคือ แชร์เมมโมรี เซกเมนต์

ถูกติดต่อทั้งการอ่านและการเขียนโดยการเรียก โปรเซสค่า SHM RDONLY จะถูกเจาะจงลงไป โดย shmflag จะให้การเข้าถึงเป็น “read only”

shmdt system call

เมื่อโปรเซสเสร็จสิ้นจาก แชร์เมมโมรี เซกเมนต์ มันจะปลดเซกเมนต์ออกโดยเรียกว่า shmdt system call

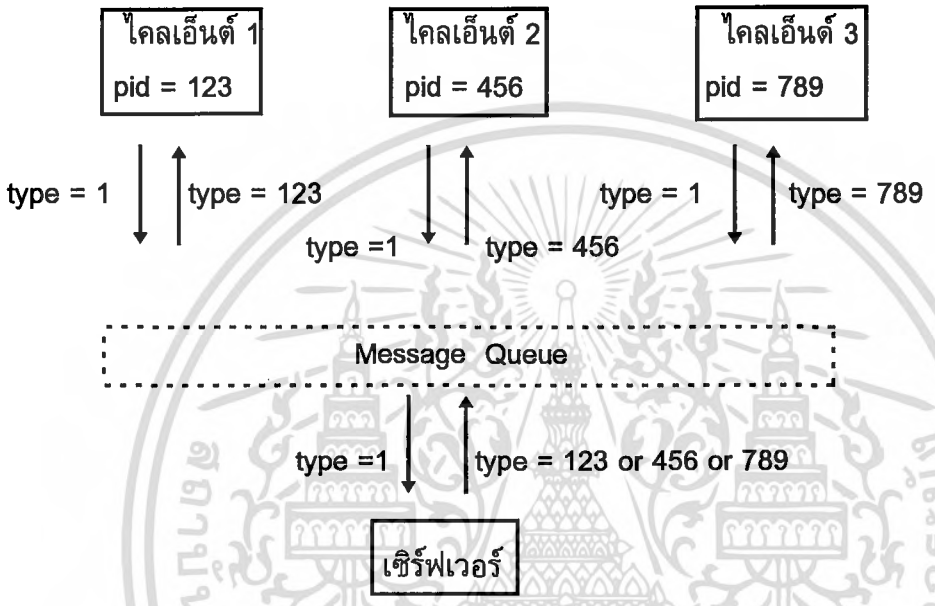
shmctl system call

shmdt system call ไม่สามารถเอา แชร์เมมโมรี เซกเมนต์ ออกได้ กรณีต้องการลบแชร์เมมโมรีเซกเมนต์ จะต้องใช้ shmctl system call

MULTIPLEXING MESSAGES

จุดประสงค์ที่มีชนิดของข้อความเป็นรูปแบบเฉพาะของตัวเองเพื่อให้มีหลายๆ โปรเซสสำหรับข้อความที่หลากหลายในคิวแบบคิวเดียว

พิจารณาเซิร์ฟเวอร์ที่มีไคลเอ็นต์หลายๆตัว เราใช้ชนิดของข้อมูลเป็น 1 สำหรับข้อความที่ถูกส่งจากไคลเอ็นต์ไปยังเซิร์ฟเวอร์ ถ้าไคลเอ็นต์ส่งหมายเลขโปรเซสของตัวเองไปกับข้อความที่ส่งด้วยแล้ว เซิร์ฟเวอร์ก็สามารถที่จะส่งข้อความไปยังไคลเอ็นต์ได้โดยใช้หมายเลข โปรเซสของไคลเอ็นต์เป็นชนิดของข้อความ



รูปที่ 2.7 แสดงถึงวิธีที่คิวเดียวสามารถใช้ในการรวมข้อความระหว่างหลายๆ โปรเซส

ลักษณะอื่น ๆ ที่ถูกกำหนดโดยชนิด.ของคุณสมบัติของข้อความ ถ้าความสามารถในการอ่านข้อความในลำดับที่ซับซ้อนกว่า First In First Out . โดยวิธีของ pipes และ FIFOs ข้อมูลจะต้องถูกอ่านตามลำดับเดียวกับที่ถูกเขียนขึ้นมา แต่ใน Message Queue เราสามารถอ่านข้อความที่ลำดับใดๆ ที่ประกอบด้วยค่าซึ่งเรากำหนดโดยชนิดของข้อความ เราสามารถกำหนดระดับความสำคัญของข้อความโดยกำหนดระดับไปยังชนิดของข้อความ หรือช่วงของชนิดของข้อความ ยิ่งไปกว่านั้นเราสามารถเรียกใช้ msgrcv ด้วยแฟล็ก IPC_NOWAIT เพื่อที่จะอ่านข้อความใดๆ ซึ่งถูกระบุชนิดของข้อความโดยคิว และคืนค่าเมื่อไม่มีชนิดของข้อความที่ระบุ

บทที่ 3

การออกแบบและการเขียนโปรแกรม

การเขียนโปรแกรมในระบบยูนิกซ์ส่วนมากจะมีฟังก์ชัน(system call)ไว้สนับสนุนการเขียนโปรแกรมดังที่กล่าวมาแล้วก่อนหน้านี้ แต่การทำโปรแกรมต้นแบบนี้ยังจำเป็นที่จะต้องศึกษาเทคนิคหรือกลวิธีเพิ่มเติม เพื่อที่จะได้โปรแกรมที่ต้องการ

จากการที่โปรแกรมนี้ต้องทำงานในระบบผู้ใช้หลายคน (multiuser) โดยแต่ละคนจะเรียกใช้โปรแกรมนี้ จึงจำเป็นต้องมีการติดต่อสื่อสารกันโดยผ่านระบบเครือข่าย โดยระบบยูนิกซ์ นั้นสนับสนุนงานดังกล่าวโดยมองโปรแกรมที่ถูกเรียกใช้นี้เป็น โปรเซสดังที่ได้กล่าวมาแล้ว ในโปรแกรมนี้เมื่อถูกเรียกใช้ 1 ครั้งหรือ 1 คนจะแบ่งตัวเองออกเป็น 2 โปรเซสทำหน้าที่ต่างกันควบคู่กันไป โดยโปรเซสหนึ่งทำหน้าที่ส่งสัญญาณหรือข้อความให้กับโปรเซสอื่นๆ ส่วนอีกโปรเซสหนึ่งจะคอยรับสัญญาณหรือข้อความจากโปรเซสอื่นๆ โดยโปรเซสหนึ่งทำหน้าที่ของตนไปโดยไม่ยุ่งเกี่ยวกัน แต่จะติดต่อถึงกันโดยใช้วิธี Share memory และ Message queue(เป็นวิธีการของการติดต่อสื่อสารระหว่างโปรเซส) โดย Shared memory จะช่วยในการใช้ตัวแปรร่วมกัน(ตามปรกติแต่ละโปรเซสจะมีตัวแปรของตัวเอง แม้ว่าจะชื่อเดียวกันก็ตาม) และใช้ message queue ในการส่งสัญญาณควบคู่จากโปรเซสแม่ไปยังโปรเซสลูก เช่น สั่งให้โปรเซสลูกจบการทำงาน เป็นต้น

การติดต่อระหว่างผู้ใช้จะใช้วิธี Message queue เพียงอย่างเดียวเพื่อส่งข้อความถึงกัน โดยผู้ใช้คนหนึ่งจะส่งข้อความผ่าน Messge queue เพียงอย่างเดียวเพื่อส่งข้อความถึงกันโดยผ่านโปรเซสแม่ของตนเอง แล้วจะคอยรับข้อความจากคนอื่นผ่าน Message queue ในการทำงาน โปรเซสลูก

ลักษณะของข้อมูลที่ส่งผ่าน Message queue จะมีลักษณะดังนี้

send	recieve	sign	text

รูปที่ 3.1 แสดงลักษณะของข้อมูล

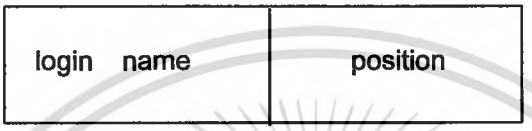
send คือ หมายเลขของผู้ส่ง โดยในโปรแกรมนี้จะมีการกำหนดหมายเลขของผู้ใช้แต่ละคนขึ้นมา ใหม่ ตามตำแหน่งของผู้เข้าร่วมประชุม เช่น 1 แทน ประธาน ,2 แทน เลขาฯ และ 3 - 12 แทนผู้เข้าร่วมประชุมคนอื่นๆ

recieveคือ หมายเลขของผู้ที่รับข้อความ ซึ่งโปรเซสลูกของผู้ใช้แต่ละคนจะอ่านข้อความเฉพาะที่มี หมายเลขส่วนนี้ตรงกับหมายเลขของผู้ใช้ของตนเองเท่านั้น

sign คือ สัญญาณที่จะส่ง เพื่อใช้ติดต่อหรือควบคุมการประชุม

text คือ ข้อความการประชุม

ในการประชุมนี้เริ่มต้นจะให้เลขาฯการประชุม (กำหนดโดย Administrator) เป็นคนเลือกผู้ที่จะเข้าร่วมประชุม และกำหนดตำแหน่งของผู้เข้าประชุมนั้นๆแล้ว เก็บลงไฟล์องค์ประชุม จากนั้นเมื่อผู้ใช้คนอื่นๆ เรียกโปรแกรมนี้ก็จะให้ไปอ่านในไฟล์ดังกล่าว ถ้าพบชื่อและตำแหน่งของตนก็สามารถเข้าประชุมได้ ในการประชุมจะใช้หลักเหมือนการประชุมจริงโดยมีประธาน เลขาฯ ตลอดจนถึงมีการบันทึกผลการประชุมลงไฟล์



รูปที่ 3.2 ลักษณะของเรคอร์ดในไฟล์องค์ประชุม

login name คือ ชื่อผู้เข้าประชุมโดยใช้ตัวเดียวกันกับของระบบ Unix โดยผ่านทาง login

position คือ ค่าตำแหน่งของคนๆนั้น เหมือนกับค่า send และ recive ที่กล่าวมาข้างต้น

รูปที่ 3.3 แสดงลักษณะของเรคอร์ดในไฟล์ผลการประชุม

text คือ ข้อความที่ผู้ใช้ที่มีชื่อตาม login name เขียนขึ้น

การทำงานส่วนเจ้าหน้าที่(Administrator)

โปรแกรมระบบต้นแบบการประชุมอิเล็คทรอนิกส์จะสามารถทำงานได้ ต้องมีส่วนที่จะต้องกำหนดกลุ่มการประชุมและเลขาฯประจำของแต่ละกลุ่มเพื่อที่เลขาจะได้ทำการกำหนดผู้ที่สามารถเข้าร่วมประชุมได้ และนอกจากนี้เลขาฯของกลุ่มก็จะเป็นผู้กำหนดประธานกลุ่มอีกด้วย โดยรายละเอียดเหล่านี้จะเก็บไว้ในไฟล์ grpdtl.txt

โดยในไฟล์ grpdtl.txt มีโครงสร้างดังนี้

องค์ประกอบของฟิลด์

Grp.Gname เก็บข้อมูล คือ ชื่อของกลุ่มการประชุม

Grp.Sname เก็บข้อมูล คือ ชื่อของเลขาฯประจำกลุ่มการประชุม

Grp.Key เก็บข้อมูล คือ รหัสที่ใช้ในการเข้าถึง Message queue

โดยเจ้าหน้าที่(Administrator)สามารถในการกระทำกับกลุ่มดังนี้

1.สามารถดูรายละเอียดของกลุ่มการประชุม

2.สามารถทำการเพิ่มกลุ่มการประชุม

3.สามารถทำการลบกลุ่มการประชุม

4.สามารถทำการเปลี่ยนแปลงฟิลด์ต่างๆของกลุ่มเช่น รายชื่อกลุ่ม รายนามเลขาฯ รหัสในการเข้าถึง Message queue

5.สามารถทำการเปลี่ยนแปลงรหัสผ่านของการใช้โปรแกรม Administrator

ในส่วนของการเก็บรหัสผ่าน จะเก็บรหัสไว้ในไฟล์ชื่อ Pwdfile.txt

บทที่ 4

สรุปผลการวิจัยและข้อเสนอแนะ

4.1 ผลการวิจัย

งานวิจัยในการสร้างระบบต้นแบบการประชุมอิเล็คทรอนิกส์ เป็นระบบต้นแบบที่สามารถมีองค์ประชุมได้มากที่สุด 12 องค์ประชุม โดยเป็นระบบช่วยอำนวยประโยชน์สำหรับหน่วยงานและองค์กรต่างๆ ในกรณีที่ต้องการประชุม ปรึกษาหารือในด้านการบริหารงาน โดยระบบจะช่วยประหยัดเวลาที่ต้องเสียในการเดินทางเพื่อร่วมเข้าประชุม และเพื่อความคล่องตัวในการดำเนินงานต่างๆ และนอกจากนี้เนื้อหาที่ทำการประชุมยังสามารถนำมาทบทวนและไดร่ดรองอีกครั้งหนึ่งได้ด้วย

4.2 สรุปผลการวิจัย

ผลที่ได้จากการพัฒนาระบบต้นแบบการประชุมอิเล็คทรอนิกส์ มีความสามารถสรุปได้ดังนี้

1. ระบบนี้สามารถทำการประชุม โดยองค์ประชุมที่สามารถเข้าร่วมประชุมได้มากที่สุด 12 องค์ประชุม

2. ระบบต้นแบบ จะกำหนดให้เลขาฯของแต่ละกลุ่มการประชุมเป็นผู้กำหนดองค์ประชุมของแต่ละกลุ่มทั้งหมด

3. ระบบต้นแบบ จะให้เจ้าหน้าที่ทำหน้าที่กำหนดรายนามของเลขาฯในแต่ละกลุ่มการประชุม

4. ระบบต้นแบบ สามารถดูรายละเอียดเนื้อหาการประชุม ในขณะทำการประชุมและหลังจากทำการประชุมเสร็จสิ้นแล้ว

5. ระบบมีส่วนที่ทำหน้าที่ป้องกันผู้อื่นที่ไม่ใช่องค์ประชุมเข้าประชุม โดยตรวจสอบจากไฟล์การประชุม

6. กรณีองค์ประชุมต้องการออกจากการประชุมชั่วคราว ก็จะมีระบบในการป้องกันไม่ให้ผู้อื่นเข้าประชุม โดยให้มีการดั้งพาสเวอร์ดก่อนออกจากการประชุมชั่วคราว

ข้อจำกัด

1. ผู้ที่จะทำการสร้างระบบต้นแบบการประชุมอิเล็คทรอนิกส์จะต้องมีการวางแผนอย่างรัดกุม เพราะการออกแบบโครงสร้างของระบบเลียนแบบการประชุมจริง ต้องมีความละเอียด เข้าใจถึงรูปแบบการประชุมอย่างถ่องแท้

2. ระบบต้นแบบการประชุมอิเล็คทรอนิกส์ใช้สำหรับกรณีองค์ประชุมที่สามารถเข้าร่วมประชุมได้มากที่สุด 12 องค์ประชุม

3. ไม่สามารถใช้ภาษาไทยในการประชุม

4. สามารถใช้งานได้เฉพาะบนเทอร์มินอล (Terminal) vt100 เท่านั้น

4.3 แนวทางการวิจัยและข้อเสนอแนะ

ปัญหาพิเศษในหัวข้อการวิจัยระบบต้นแบบการประชุมอิเล็คทรอนิกส์ เป็นการศึกษาเพียงส่วนหลักๆ บางส่วนเท่านั้น เนื่องจากระยะเวลาในการทำปัญหาพิเศษมีค่อนข้างจำกัด ดังนั้นจึงขอเสนอเพื่อให้ผู้ที่มีความสนใจในปัญหาพิเศษหัวข้อนี้ ได้ทำการศึกษาและพัฒนาต่อไปนี้ดังนี้

1. ข้อจำกัดในกรณีองค์ประชุมที่ในระบบต้นแบบสามารถเข้าร่วมประชุมได้มากที่สุด 12 องค์ประชุม ให้สามารถเข้าร่วมประชุมได้มากกว่านี้

2. การจัดการหน้าจอการทำงาน สามารถพัฒนาเพื่อให้เกิดความคล่องตัวมากขึ้น

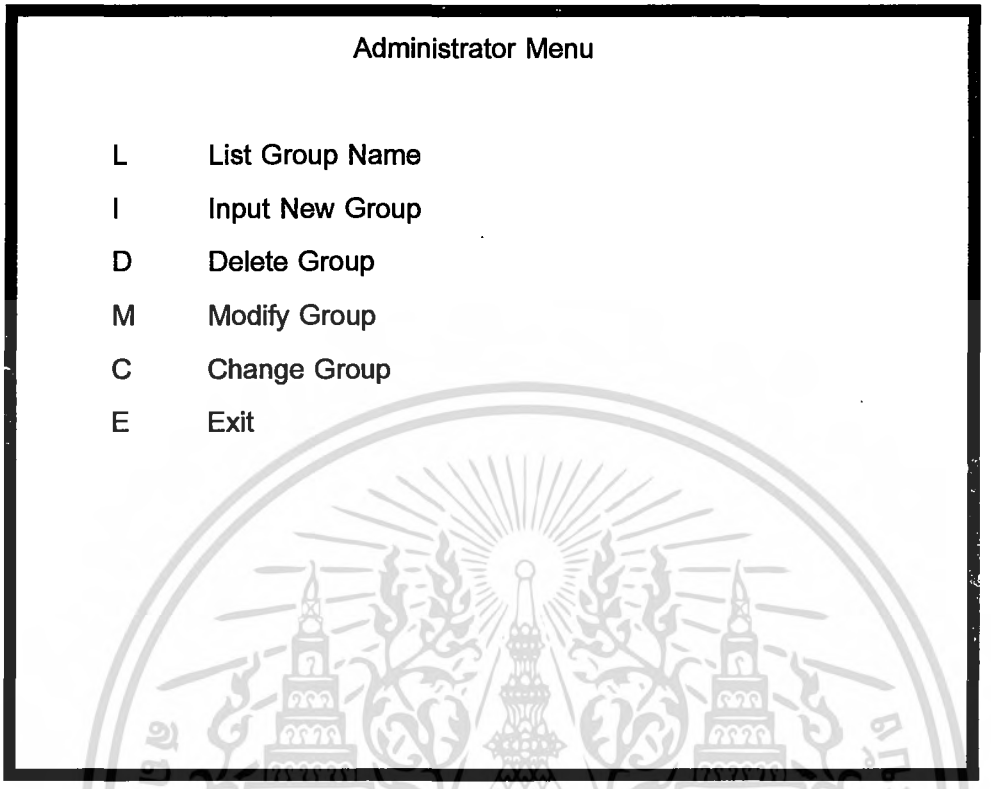
3. สามารถใช้ภาษาไทยในการประชุมได้

4. สามารถใช้งานได้ที่เทอร์มินอล (Terminal) หลายๆชนิด

ภาคผนวก ก

ลักษณะหน้าจอ และการใช้งาน

ลักษณะหน้าจอเจ้าหน้าที่ (Administrator Menu)



ส่วนประกอบของหน้าจอเจ้าหน้าที่

L List Group Name : กด คีย์ L เพื่อดูรายละเอียดข้อมูลของกลุ่มต่างๆ ได้แก่ ชื่อกลุ่ม ชื่อเลขาฯ และ รหัสคีย์

I Input New Group : กด คีย์ I เพื่อเพิ่มกลุ่ม

D Delete Group : กด คีย์ D เพื่อลบกลุ่ม

M Modify Group : กด คีย์ M เพื่อเปลี่ยนแปลงรายละเอียดของกลุ่ม

C Change Password : กด คีย์ C เพื่อเปลี่ยนพาสเวอร์ด

E Exit : กด คีย์ E เพื่อออกจากโปรแกรม

วิธีการใช้งาน โปรแกรม Admin

1.เมื่อกดคีย์ L (List) เพื่อขอดูรายละเอียดของกลุ่มต่างๆ จะปรากฏหน้าจอดังนี้

Order	Group Name	Secretary Name	Key
1	G1	Sec1	1234
2	G2	Sec2	2345
3	G3	Sec3	3456

Press 'M' return to Main Menu

2.เมื่อกดคีย์ I (Input)เพื่อทำการเพิ่มกลุ่มใหม่จะปรากฏหน้าจอดังนี้

Input DATA

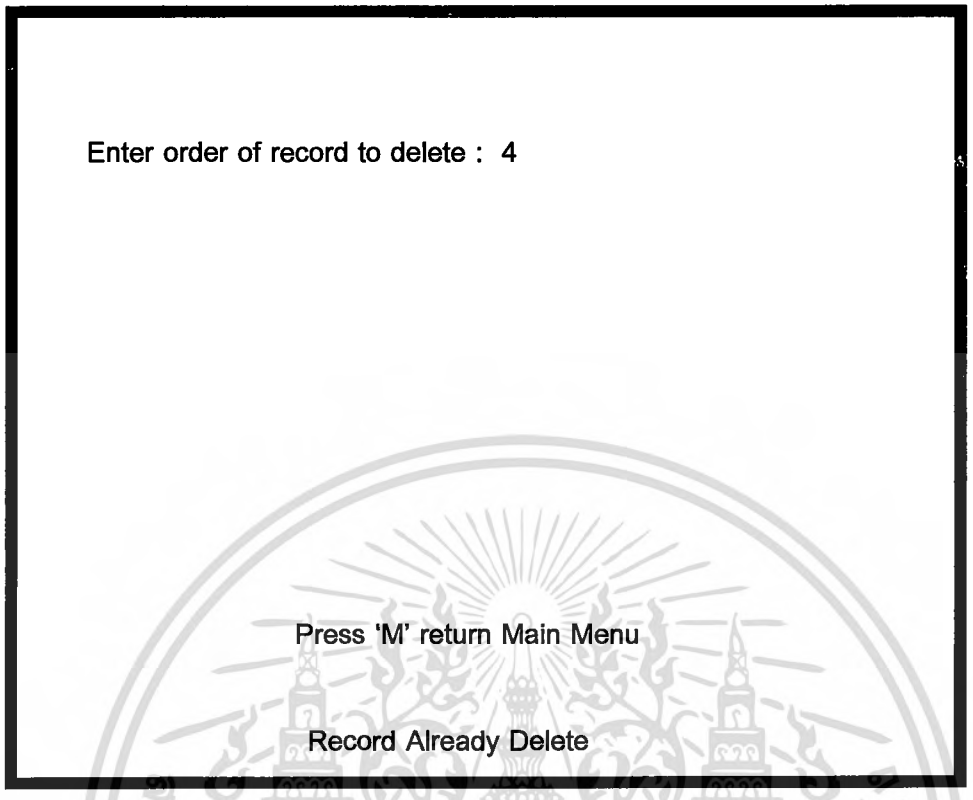
Enter Group Name : G4

Enter Secretary Name : Sec4

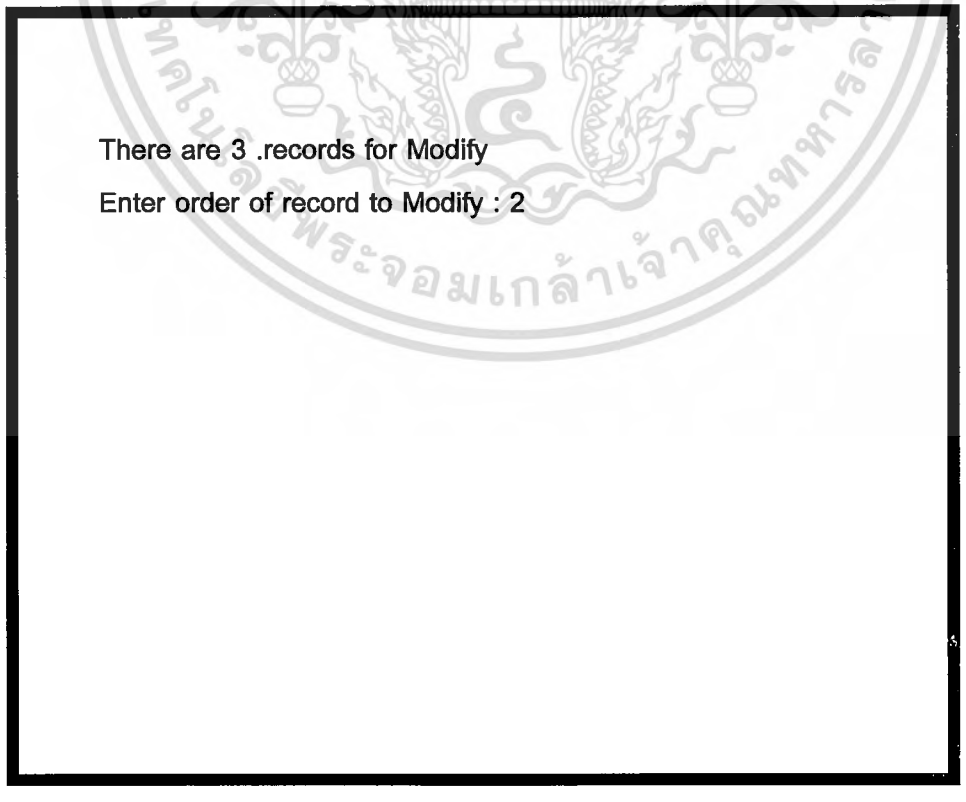
Enter Group Key : 4567

'I' Input Data 'M' Main Menu

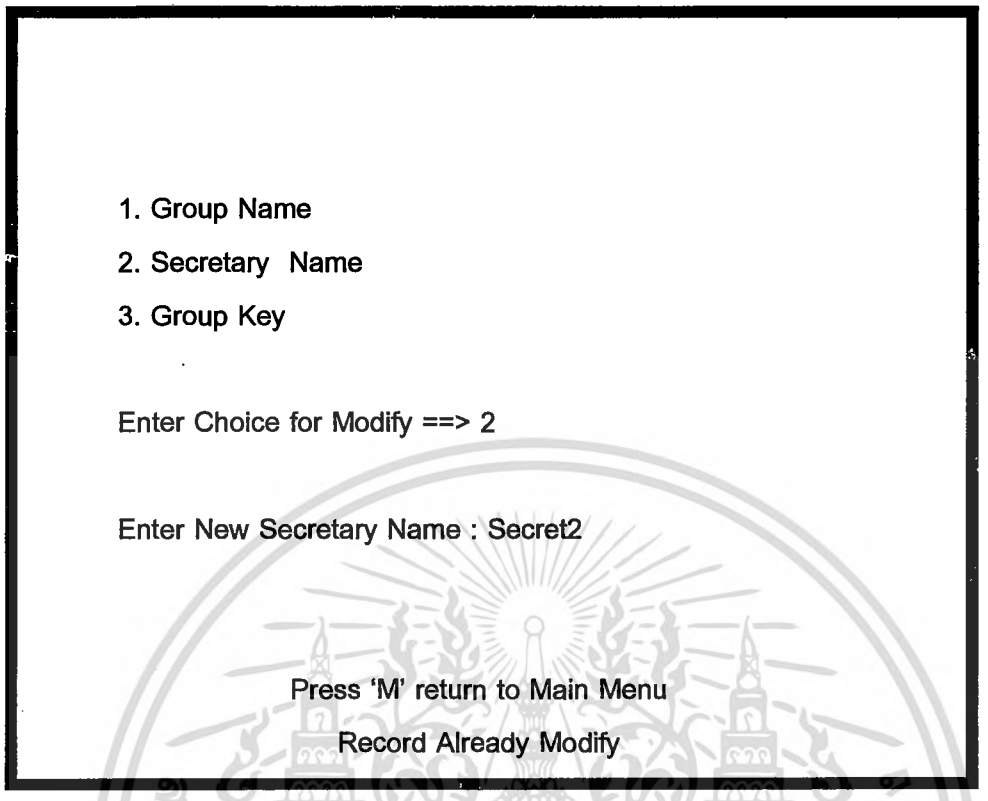
3.เมื่อกดคีย์ D (Delete)เพื่อทำการลบกลุ่ม จะปรากฏหน้าจอดังนี้



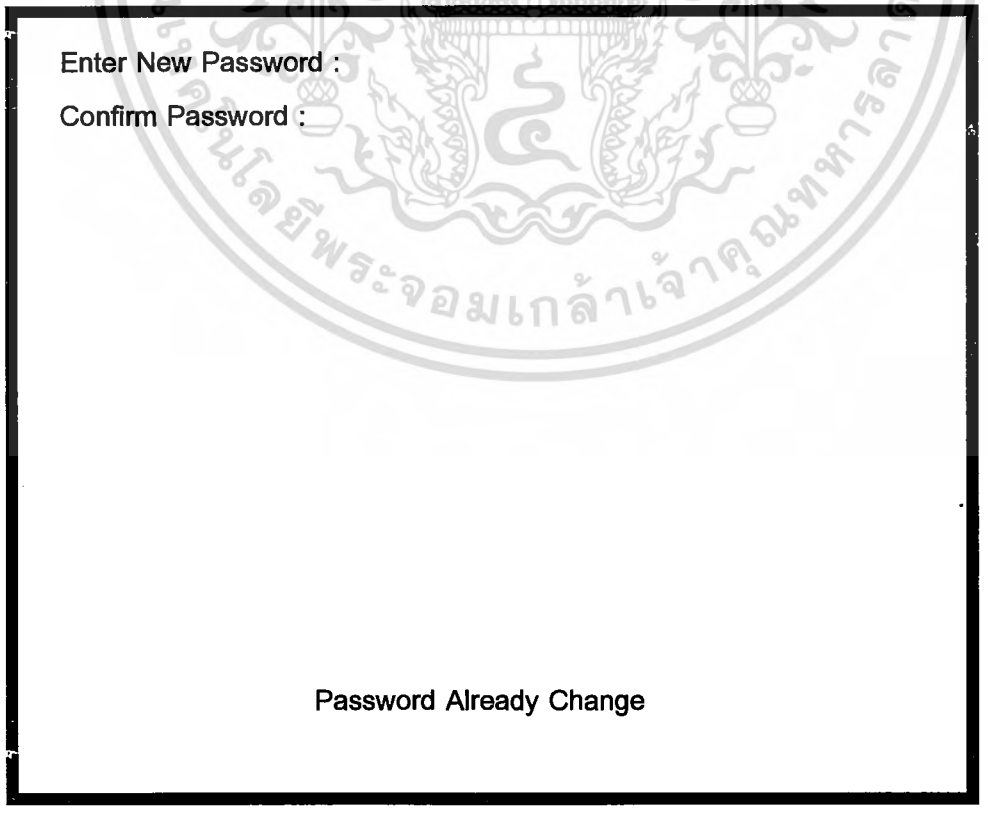
4. เมื่อกดคีย์ M(Modify)เพื่อทำการเปลี่ยนแปลงข้อมูลกลุ่ม จะปรากฏหน้าจอดังนี้



และจะปรากฏหน้าจอเพื่อเลือกว่าต้องการเปลี่ยนแปลงฟิลด์ใด ตามเลขที่กำกับ



5.เมื่อกดคีย์ C (Change)เพื่อทำการเปลี่ยนแปลง พาสเวอร์ด จะปรากฏหน้าจอดังนี้



ลักษณะหน้าจอประธาน

YOUR NAME : 1)	USER LIST	MODE:<<.....>>
SECRETARY : 2)	3)	4)
I : talk or interupt user	5)	6)
F(n) :user n Q **Q** : by queue	7)	8)
D :clear queue A : Exit	9)	10)
F :conference file list	11)	12)
Current action :		Your action :
MSG :		

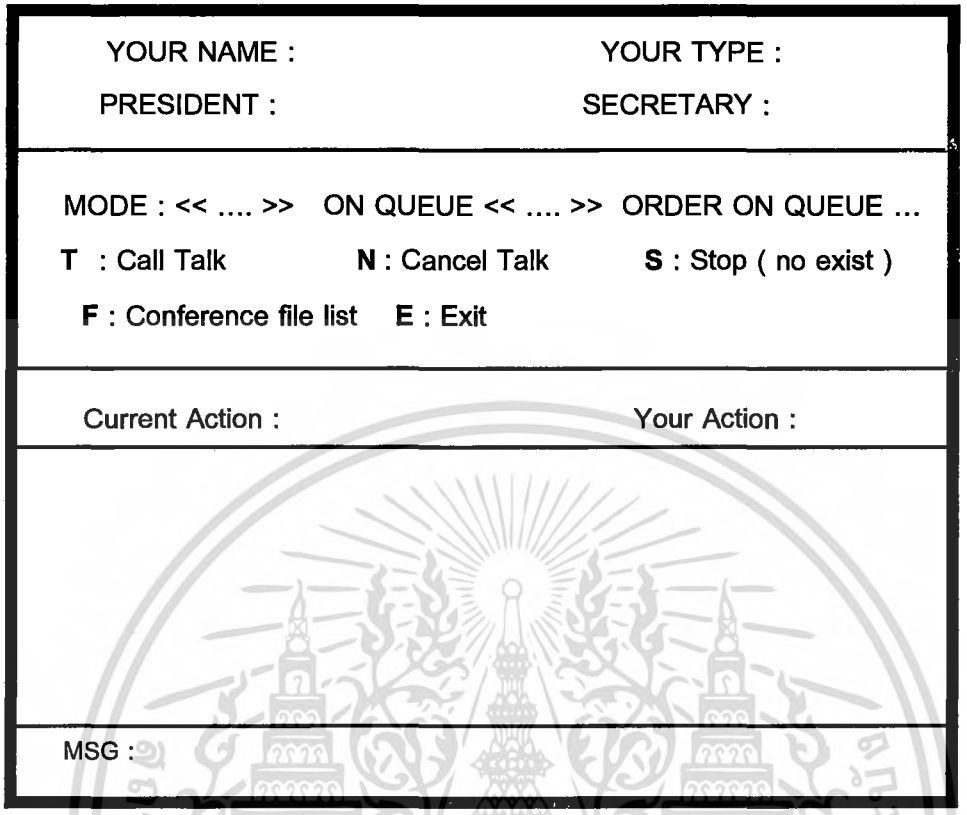
ส่วนประกอบของหน้าจอประธาน

YOUR NAME :1) ชื่อประธาน

SECRETARY :2) ชื่อเลขาฯของกลุ่ม

I:	กด key I เมื่อประธานต้องการพูด หรือต้องการพูดแบบขัดจังหวะ
F(n):	ประธานกด Fuction Key เพื่ออนุญาติให้องค์ประชุมที่ประธานต้องการให้ออกความเห็นคีย์ข้อความ โดยดูรายนามองค์ประชุมจาก USER LIST
Q:	กด key Q เมื่อประธานต้องการอนุญาติให้องค์ประชุมออกความเห็นดามลำดับการเรียก (queue)
D:	กด key D เมื่อประธานต้องการยกเลิกลำดับการเรียก (clear queue)
A:	กด key A เมื่อประธานต้องการจบการประชุม
F:	กด key F เมื่อประธานต้องการดูรายละเอียดของการไฟล์การประชุม
USER LIST :	ส่วนแสดงรายนามขององค์ประชุมดามลำดับการเข้าประชุม
MODE:	แสดงรูปแบบการทำงาน กรณี MODE:<< control >>สำหรับการทำงานแบบควบคุม กรณี MODE:<<message>>สำหรับการทำงานแบบคีย์ข้อความ
Current action:	ส่วนแสดงข้อความในการทำงานต่างๆ
Your action :	แสดงรูปแบบการทำงาน

ลักษณะหน้าจอองค์ประชุม



ส่วนประกอบของหน้าจอองค์ประชุม

YOUR NAME :	ชื่อผู้เข้าประชุม
PRESEDENT :	ชื่อประธาน
TOUR TYPE :	รหัสประจำตัวผู้เข้าประชุม
SECRETARY :	ชื่อเลขา
MODE:	แสดงรูปแบบการทำงาน
	กรณี MODE:<< control >>สำหรับการทำงานแบบควบคุม
	กรณี MODE:<<message>>สำหรับการทำงานแบบคีย์ข้อความ
On queue:	แสดงรูปแบบว่าผู้เข้าประชุมมีลำดับในการออกความเห็นหรือไม่
	กรณี On queue: << Y >> สำหรับกรณีมีลำดับใน queue
	กรณี On queue: << N >> สำหรับกรณีไม่มีลำดับใน queue
Order on queue:	แสดงลำดับในการออกความเห็น
T:	กด key T เมื่อผู้เข้าประชุมต้องการแสดงความคิดเห็น
N:	กด key N เมื่อผู้เข้าประชุมยกเลิกการแสดงความคิดเห็น
S:	กด key S เมื่อผู้เข้าประชุมต้องการออกจากการประชุมชั่วคราว
F:	กด key F เมื่อผู้เข้าประชุมต้องการดูรายละเอียดของการไฟล์การประชุม
E:	กด key E เมื่อผู้เข้าประชุมต้องการออกจากการประชุม
Current action:	ส่วนแสดงข้อความในการทำงานต่างๆ
Your action :	แสดงรูปแบบการทำงาน

วิธีการใช้งาน

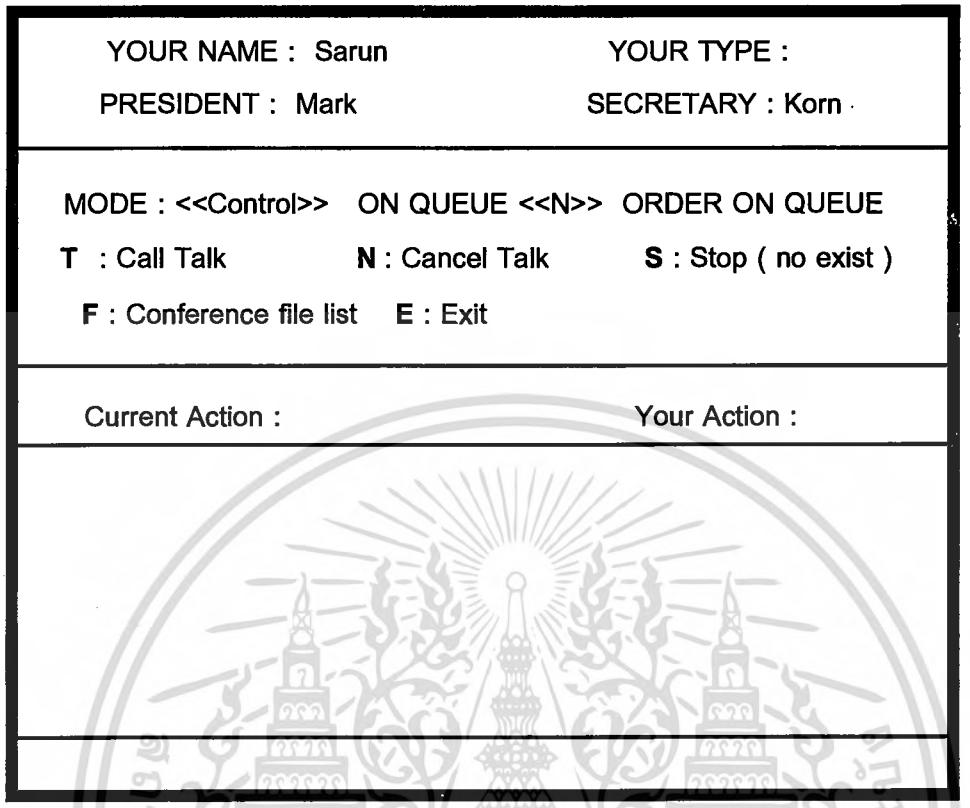
1. เมื่อเข้ามาในส่วนของการประชุม
   กรณีประธานเข้ามาก่อน จะปรากฏหน้าจอของประธานดังนี้

YOUR NAME : 1) Mark	USER LIST MODE : <<Control>>
SECRETARY : 2) Korn	3) No present 4) No present
I : talk or interupt user	5) No present 6) No present
F(n) :user n Q **Q** : by queue	7) No present 8) No present
D :clear queue A : Exit	9) No present 10) No present
F :conference file list	11) No present 12) No present
Current action :	Your action :

หลังจากนั้นเมื่อมีองค์ประชุมเข้ามา หน้าจอประธานจะปรากฏดังนี้

YOUR NAME : 1) Mark	USER LIST MODE:<< Control >>
SECRETARY : 2) Korn	3) Sarun 4) Theera
I : talk or interupt user	5) No present 6) No present
F(n) :user n Q **Q** : by queue	7) No present 8) No present
D :clear queue A : Exit	9) No present 10) No present
F :conference file list	11) No present 12) No present
Current action :	Your action :

และหน้าจอองค์ประชุมจะปรากฏดังนี้

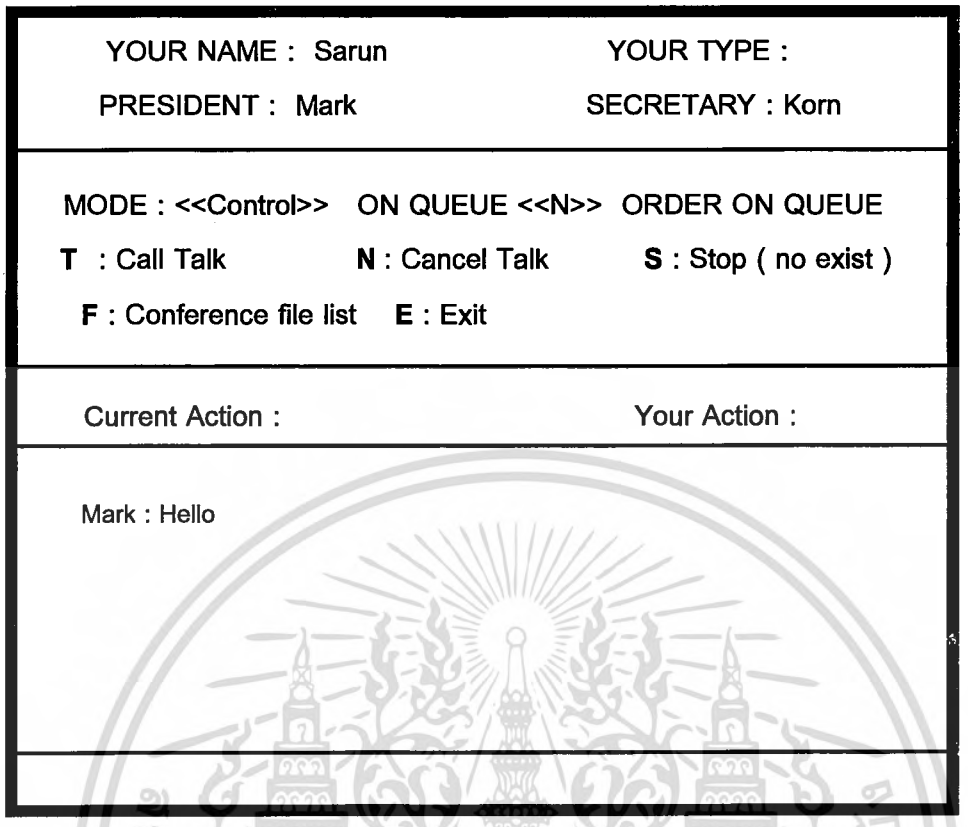


2. เริ่มต้นทำการประชุม

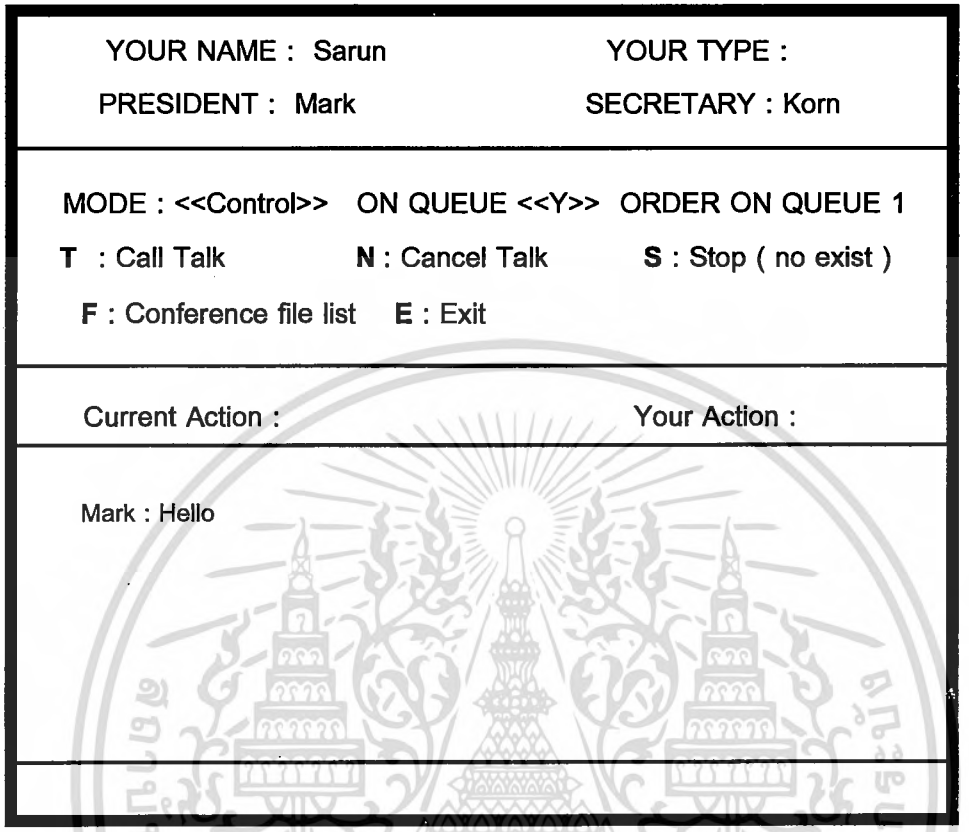
ประธานทำการเปิดประชุม และกดkey I ใน Mode Control เพื่อคีย์ข้อความ จะปรากฏหน้าจอประธานดังนี้

YOUR NAME : 1) Mark	USER LIST MODE:<< Message>>
SECRETARY : 2) Korn	3) Sarun 4) Theera
I : talk or interupt user	5) No present 6) No present
F(n) :user n Q Q : by queue	7) No present 8) No present
D :clear queue A : Exit	9) No present 10) No present
F :conference file list	11) No present 12) No present
Current action :	Your action :
Mark : Hello	
MSG : Hello	

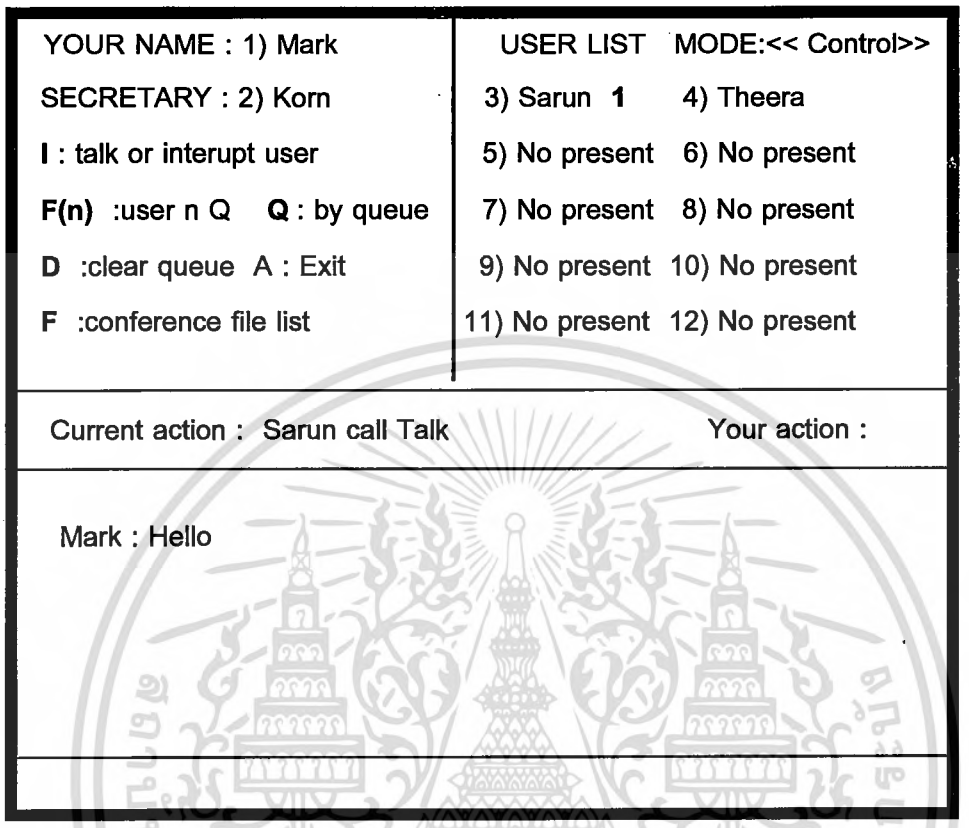
และหน้าจอองค์ประชุมหลังจากประธาน กด key I และคีย์ข้อความ จะปรากฏดังนี้



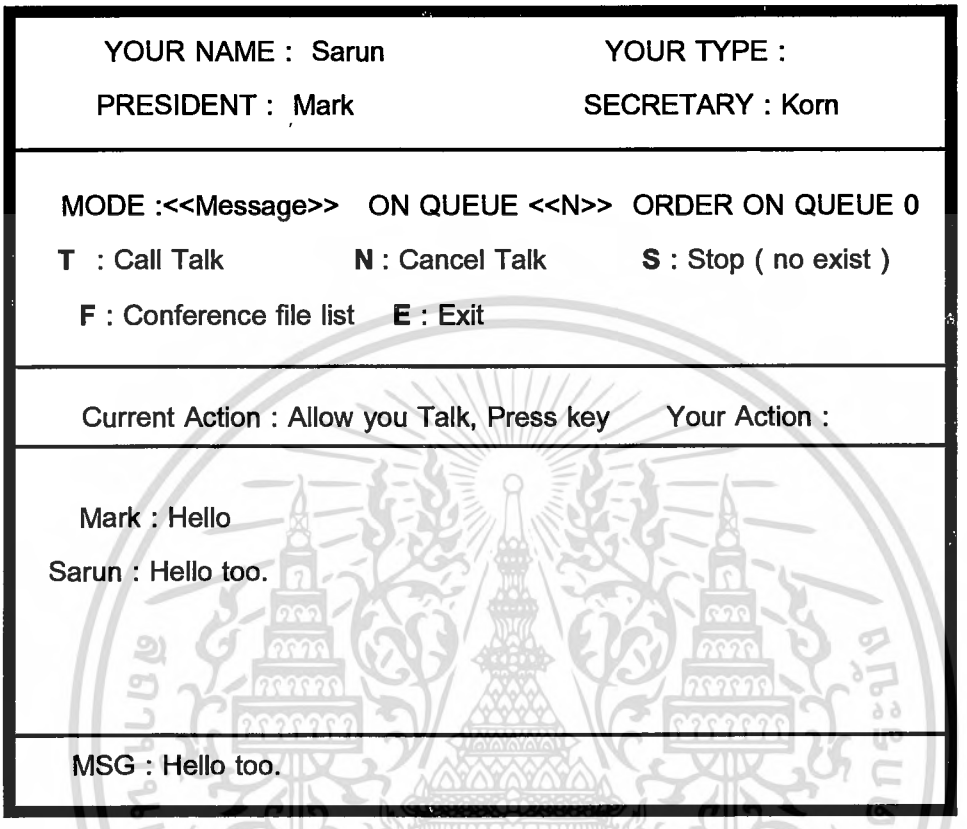
3. เมื่อผู้เข้าประชุมแสดงความคิดเห็น โดยการกด key T ใน Mode Control หน้าจอผู้เข้าประชุมปรากฏดังนี้



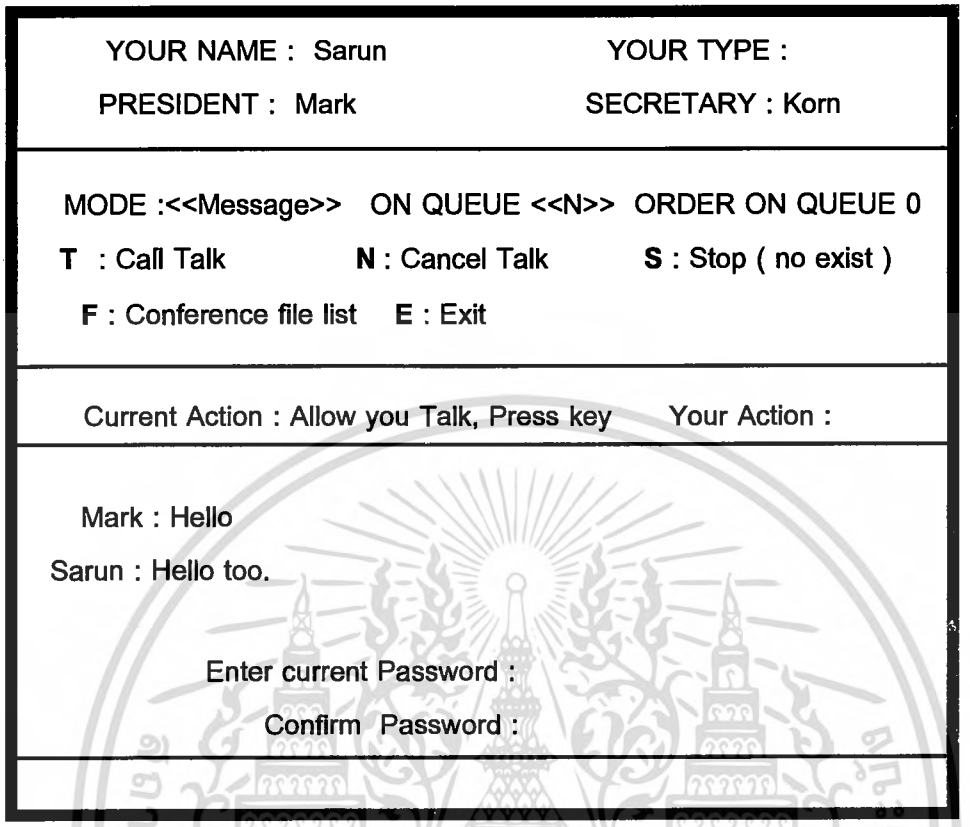
และหน้าจอประธานหลังจากองค์ประชุมต้องการแสดงความคิดเห็น เพื่อให้ประธานอนุญาติ ปรากฏดังนี้



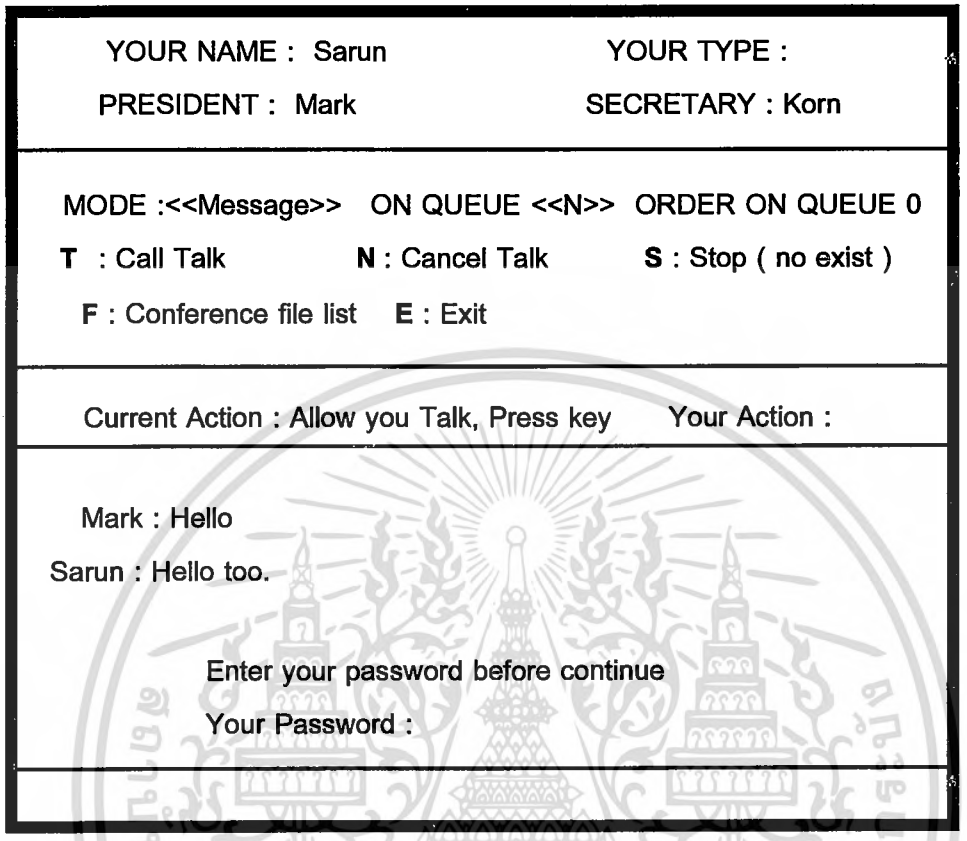
4. เมื่อประธานอนุญาติโดยการกด Function Key หรือ กด key หน้าจอขององค์ประชุม จะปรากฏดังนี้



5. เมื่อองค์ประชุมต้องการออกจากที่ประชุมชั่วคราว โดยการกด key S จะปรากฏหน้าจอให้ใส่ password เพื่อป้องกันไม่ให้ผู้อื่นเข้ามาประชุมแทน ดังนี้



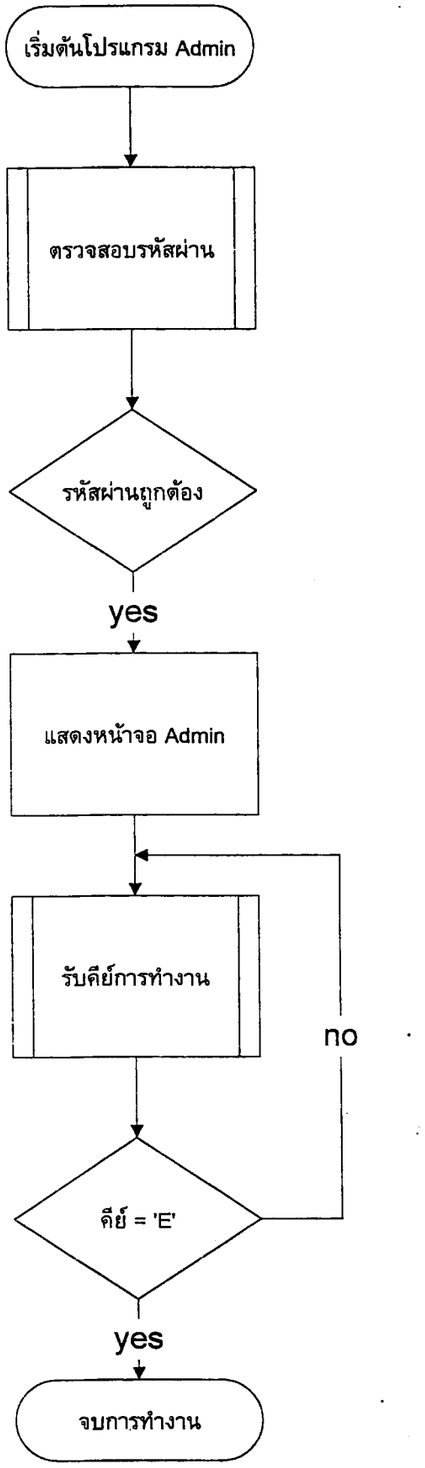
จากนั้นหน้าจอในขณะพัก เพื่อรอการใส่รหัส password จะปรากฏดังนี้

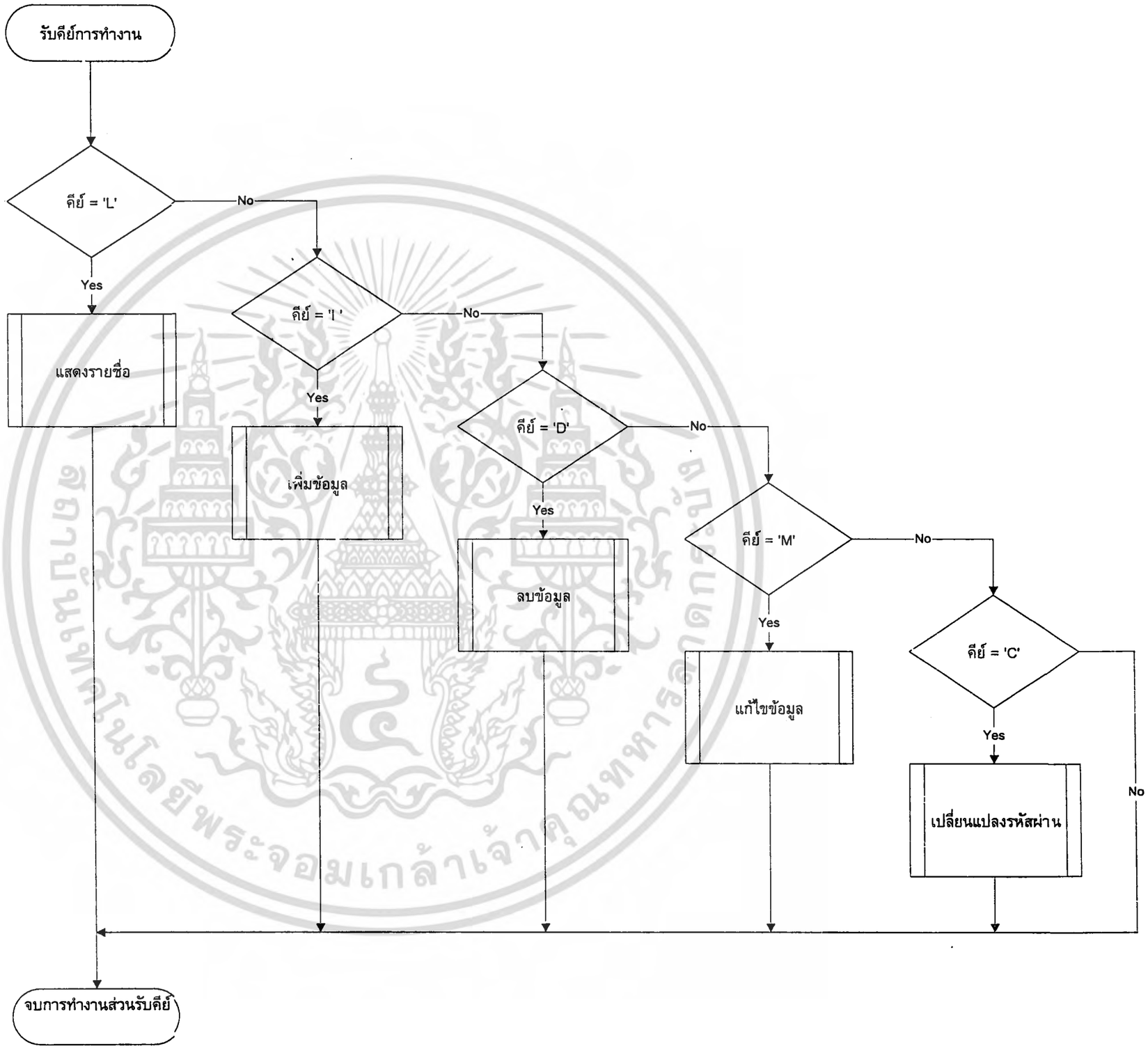


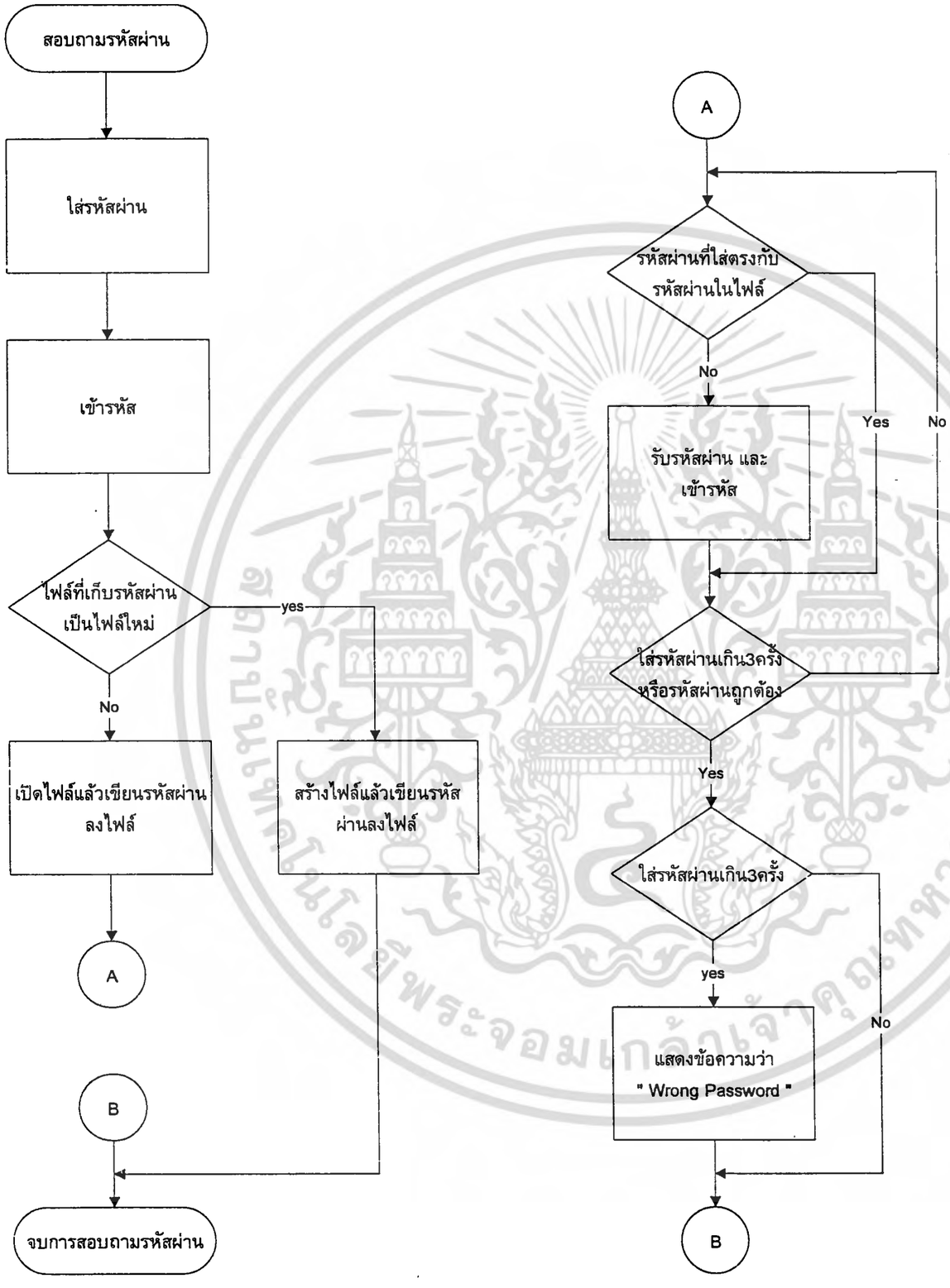
6. กรณีประธานต้องการปิดประชุม จะกด key A และจะส่งสัญญาณไปยังองค์ประชุมเพื่อรับทราบ

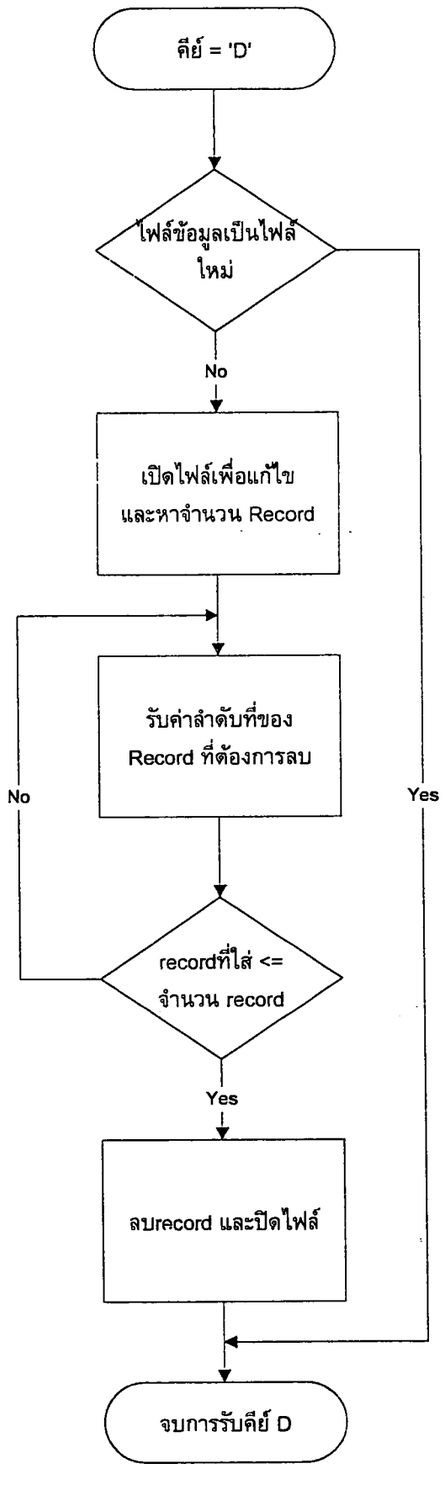
ภาคผนวก ข

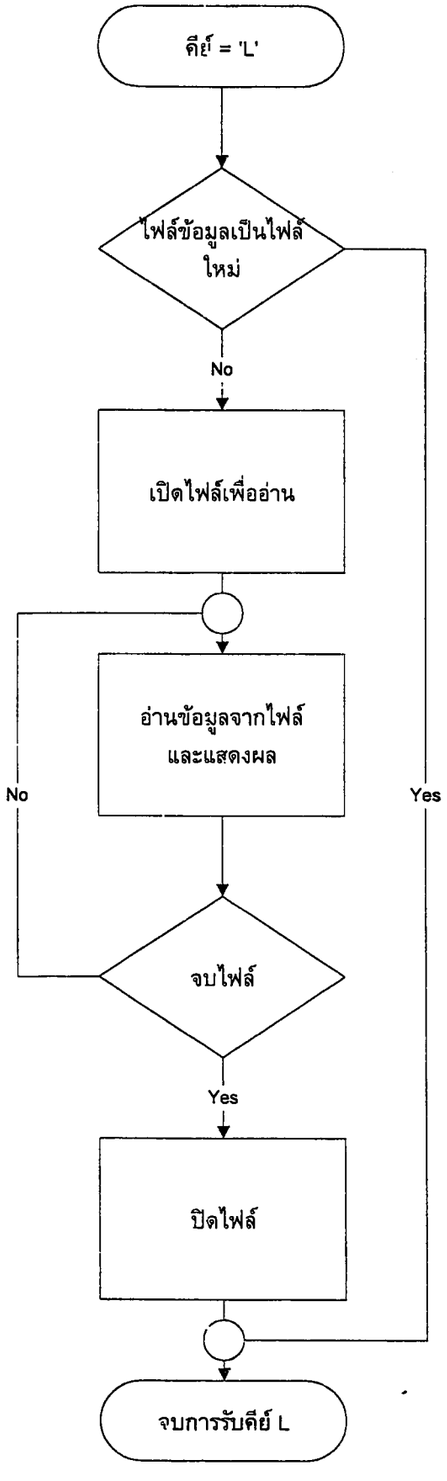
โฟล์วชาร์ตของโปรแกรม

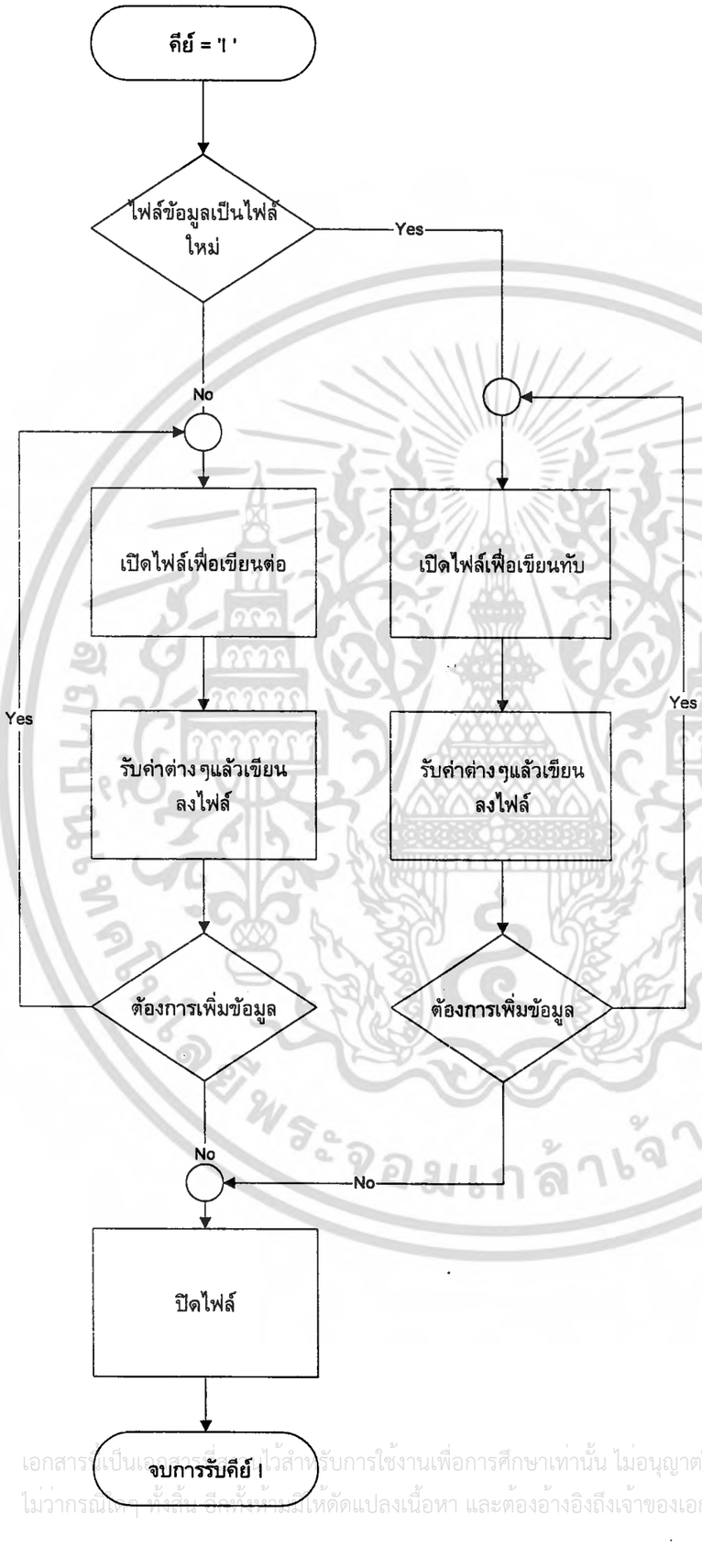


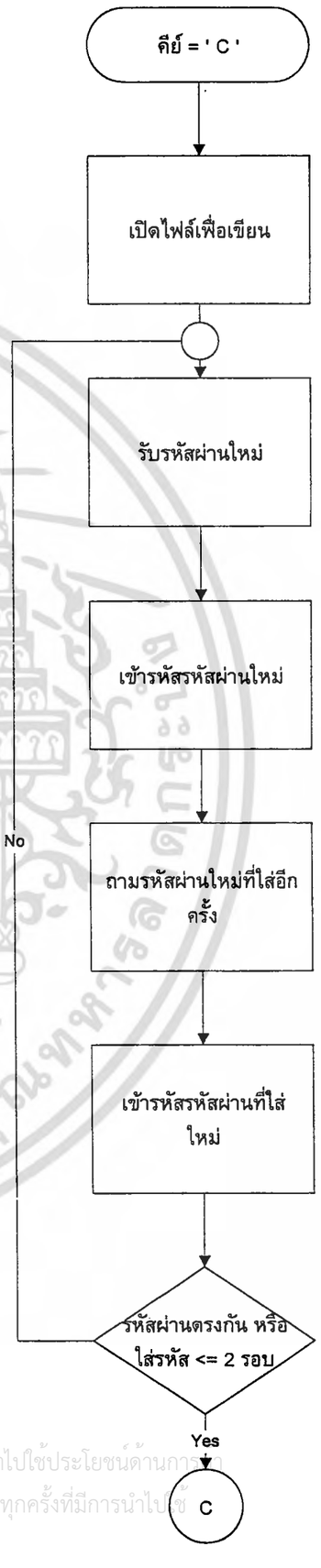


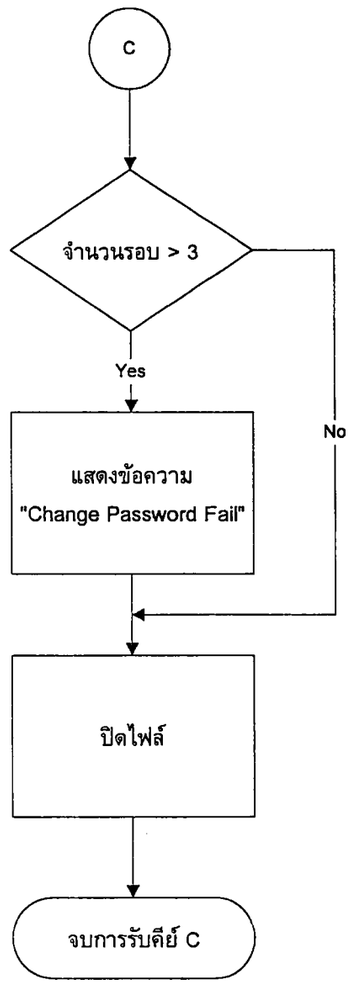


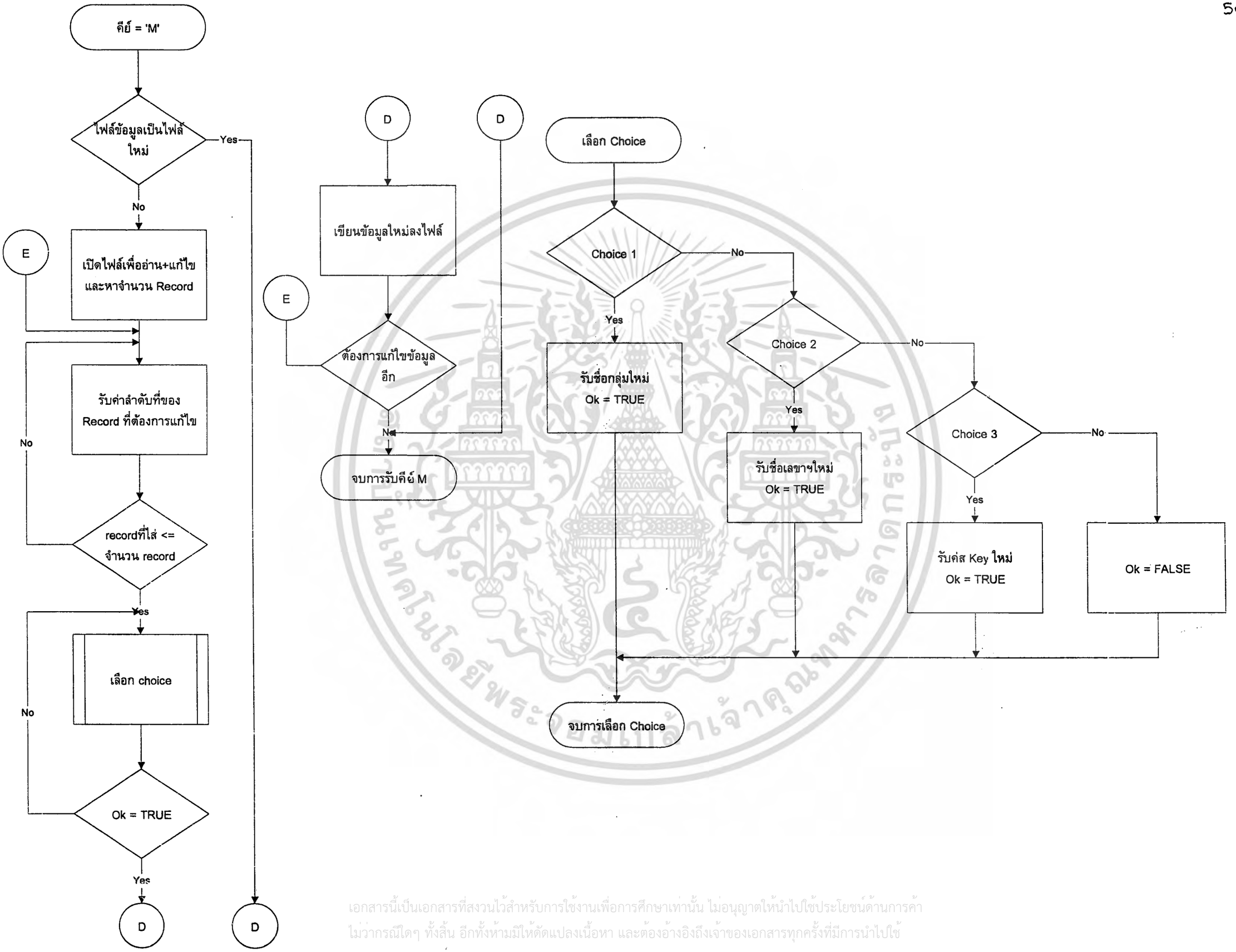


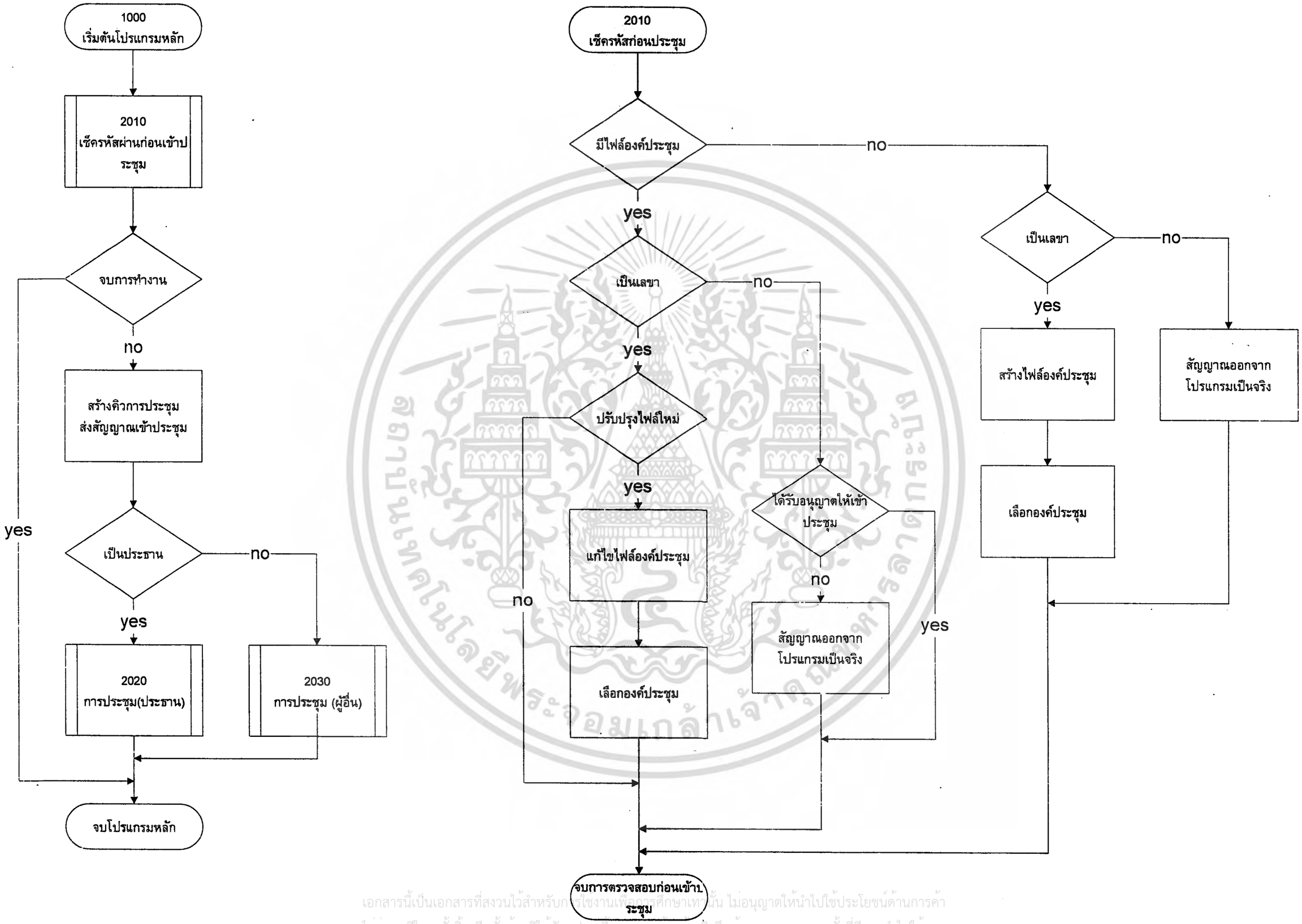


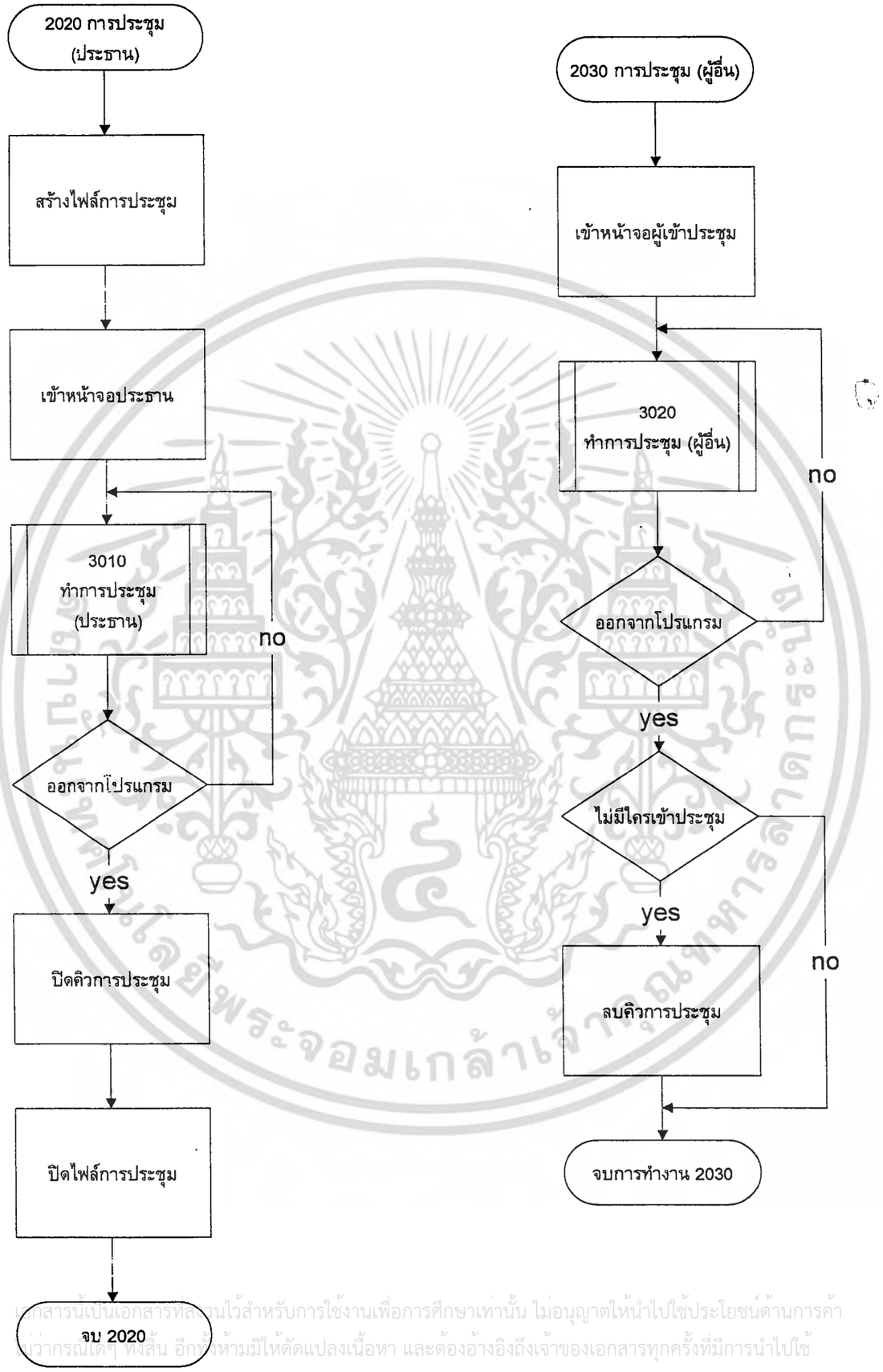


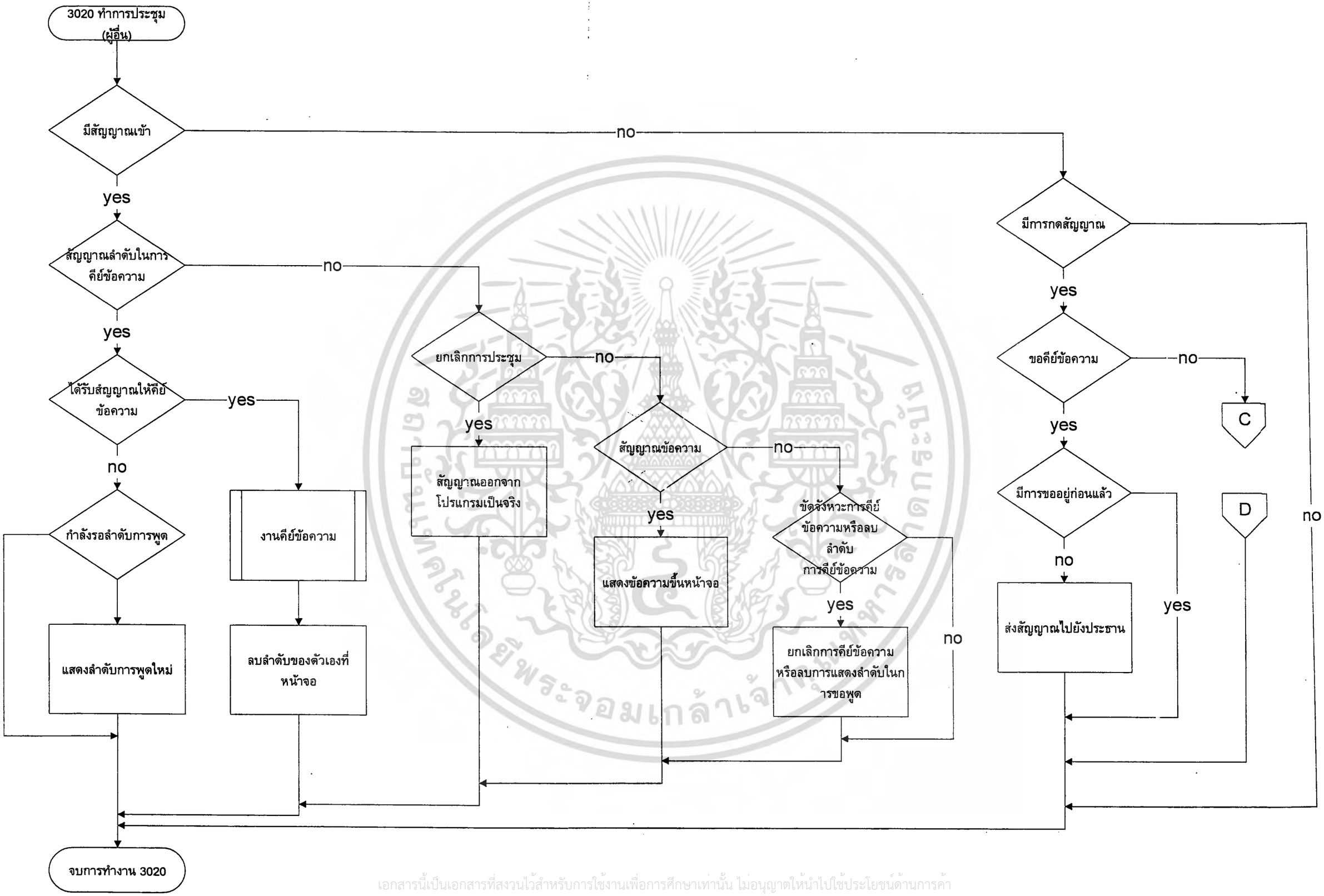


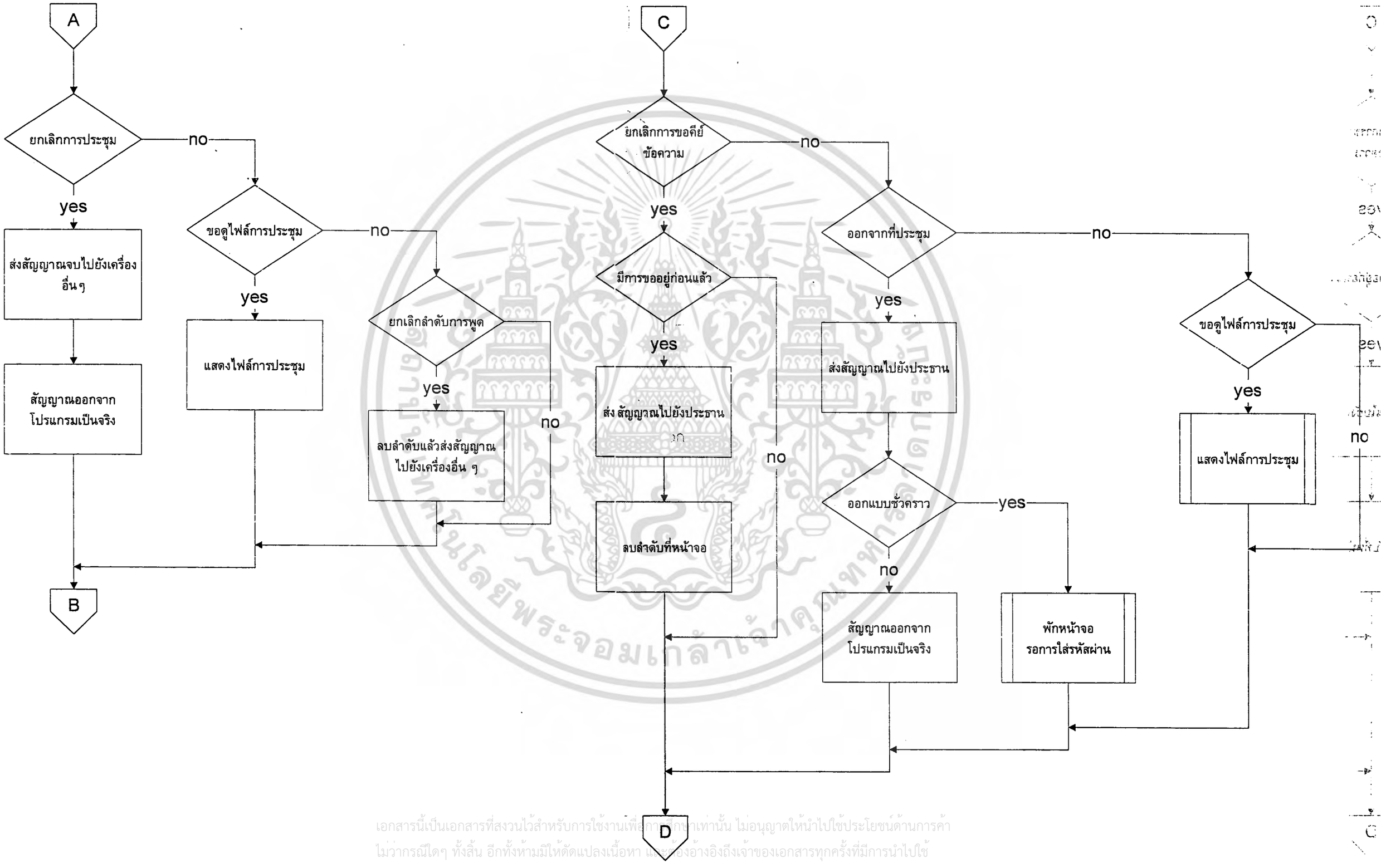


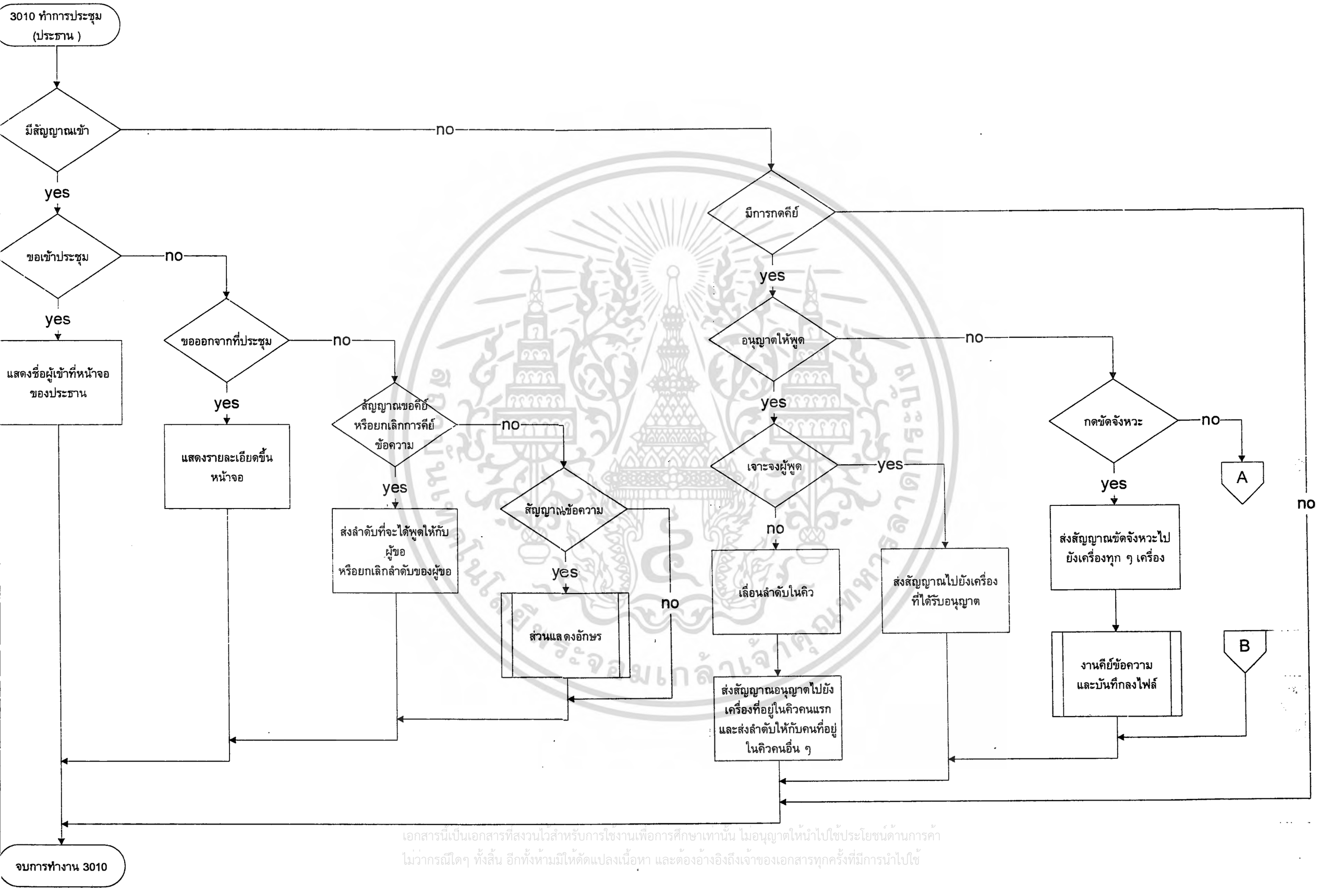












เอกสารอ้างอิง

1. Keith Haviland, Ben Salama, Unix System Programming , Second edition , Addison - Wesley Publishing ,Great Britain , 1988
2. W. Richard Stevens, Unix Network Programming , Prentice Hall , New Jersey, 1991
3. Kenneth E. Martin , C Through Unix , WCB, United Sates of America, 1992