การพัฒนาแอพพลิเคชันบนระบบยูนิกซ์โดยใช้ RPC

UNIX Network Programming using RPC

โดย

นายพรเทพ เซี่ยวโหล

นางสาวสุมณฑา หลิมศิริวงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

การพัฒนาแอพพลิเคชันบนระบบยูนิกซ์โดยใช้ RPC

UNIX Network Programming using RPC

โดย

นายพรเทพ เชี่ยวโหล 34104230

นางสาวสุมณฑา หลิมศิริวงษ์ 34108448

อาจารย์ที่ปรึกษา

อาจารย์ กฤตวัน เครือตราชู

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีประจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาแอพพลิเคชันบนระบบยูนิกซ์โดยใช้ RPC

ผู้จัดทำ

1. นายพรเทพ เซี่ยวโหล

2. นางสาวสุมณฑา หลิมศิริวงษ์

.....อาจารย์ที่ปรึกษา

(.....)

การพัฒนาแอพพลิเคชันบนระบบยูนิกซ์โดยใช้ RPC

นายพรเทพ เซี่ยวโหล
นางสาวสุมณฑา หลิมศิริวงษ์

อ. กฤตวัน เครือตราชู
ปีการศึกษา 2537

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของโครงงานวิจัยเพื่อศึกษาระบบประมวลผลแบบกระจาย (Distributed Processing) โดยใช้การพัฒนาระบบจำลองการซื้อขายหลักทรัพย์ ซึ่งมีการทำงานแบบเรียลไทม์ (real time) เป็นกรณีศึกษา โดยใช้หลักการรีโมตโพรซีเจอร์คอลลิง (RPC) เพื่อการติดต่อสื่อสารระหว่างโฮสต์ผ่านเครือข่าย

ควบคุมความสอดคล้อง (Synchronization Control) และควบคุมความถูกต้องตรงกันของข้อมูล (Consistensy Control) ระหว่างการทำงานของโพรเซสโดยใช้การส่งข่าวสารผ่านเมสเสจคิว (Message Queue) ซึ่งเป็นวิธีการหนึ่งของอินเตอร์โพรเซสคอมมิวนิเคชัน (IPC) และการตรวจสอบสถานะของข้อมูลโดยใช้แฟลกบอกการเปลี่ยนแปลง

UNIX Network Programming using RPC

Pornthep Sievlho

Sumonta Limsiriwong

Kritawan Kruatrachue

1994

Abstract

This thesis deals with the study of Synchronous Distributed Proceesing system by using a stock exchange simulation system running on a network, a real-time application, as the case-study.

This Project using Remote Procedure Call (RPC) to develop communication between processes on different hosts in UNIX system, to develop Synchronization Control and Consistency Control by Message Queues, which is one of InterProcess Communication (IPC), and chaged-flag variable.

สารบัญ

สารบัญรูป

รูป หน้าที่

สารบัญตาราง

ตาราง หน้า

บทที่ 1
บทนำ

งานพัฒนาระบบเครือข่ายกำลังกลายเป็นคำตอบที่ดีที่สุดคำตอบหนึ่ง สำหรับการพัฒนาความคุ้มค่าของระบบคอมพิวเตอร์ การพัฒนาเครือข่ายในปัจจุบันมีความสำคัญมากขึ้น นอกจากความประหยัดอันเกิดจากการใช้ทรัพยากรได้คุ้มค่าแล้ว การสื่อสารยังเป็นเครื่องมือสำหรับการแข่งขันทางธุรกิจ และความร่วมมือทางธุรกิจ

การพัฒนาเครือข่ายการทำงานแบบไคล์เอ็นต์เซอร์ฟเวอร์ก็เป็นรูปแบบหนึ่ง ที่ได้รับการพัฒนาอย่างกว้างขวาง เนื่องจากเหมาะสมกับการใช้งานได้หลากหลายประเภท เช่น งานจัดการฐานข้อมูล เป็นต้น นอกจากนี้ ยังช่วยให้เครื่องคอมพิวเตอร์จำนวนหลายเครื่องสามารถทำงานกับข้อมูลเดียวกันได้อย่างสอดคล้อง (Synchronization) และประมวลผลได้อย่างถูกต้อง (Consistance)

จากเหตุผลต่าง ๆ เหล่านี้ จึงมีส่วนผลักดันให้มีการศึกษาการพัฒนาแอพพลิเคชันที่ทำงานอยู่บนระบบปฏิบัติการยูนิกซ์ ซึ่งมีความสามารถในการทำงานหลายอย่างพร้อมกัน (multi-tasking) ให้แอพพลิเคชันที่พัฒนาสามารถทำงานอยู่บนแต่ละโฮสต์ (host) ติดต่อกับผ่านระบบเครือข่าย

1.1 วัตถุประสงค์

โครงงานนี้เป็นการศึกษาเพื่อออกแบบ และทดสอบวิธีการควบคุมความสอดคล้องในการประมวลผล และ การควบคุมความถูกต้องตรงกันของข้อมูลขณะที่ทำการประมวลผลแบบกระจาย โดยใช้ระบบจำลองการซื้อขายหลักทรัพย์เป็นกรณีศึกษา (case Study) ซึ่งเป็นตัวอย่างที่ทำให้เห็นผลการทำงานของเทคนิคที่ใช้ได้อย่างชัดเจน นอกจากนั้นยังเป็นการศึกษาวิธีการรับส่งข้อมูลผ่านเครือข่าย แอพพลิเคชันนี้พัฒนาในเท็กซ์โหมดของยูนิกซ์ และให้มีการติดต่อกับผู้ใช้แบบหน้าต่างเท็กซ์โหมด

1.2 ขอบเขตของโครงงาน

ปริญญานิพนธ์นี้จะศึกษาการพัฒนาระบบจำลองการซื้อขายหลักทรัพย์ โดยการพัฒนาให้แต่ละโฮสต์ทำงานเสมือนเป็นหน่วยงานหนึ่ง หรือ นายหน้าผู้ขายหลักทรัพย์ หรือ โบรกเกอร์ ที่ต้องการติดต่อไปยังตลาดหลักทรัพย์ซึ่งทำหน้าที่โดยอีกโฮสต์หนึ่งในเครือข่ายเช่นกัน โฮสต์แต่ละตัวที่ทำหน้าที่เป็นโบรกเกอร์จะประมวลผลต่าง ๆ เอง ก่อนที่จะส่งคำร้องขอไปยังตลาดหลักทรัพย์ เพื่อให้ประมวลผลการซื้อขายอัตโนมัติแก่รายการเสนอซื้อขายที่ตนเสนอเข้าไป หรือเพื่อร้องขอให้ตลาดหลักทรัพย์ส่งข่าวสารมาให้แก่ตนเพื่อจัดการแสดงข่าวสารแก่ผู้ใช้ด้วยฟังก์ชันอินเทอร์เฟซบนโฮสต์ของตนเอง

การทำงานบนโฮสต์ที่เป็นโบรกเกอร์ ประกอบด้วยฟังก์ชันทำงานหลัก ๆ ได้แก่ การจัดทำรายการเสนอซื้อขายหลักทรัพย์ การแสดงข้อมูลราคาเสนอซื้อขายหลักทรัพย์แก่ผู้ใช้ และการแสดงผลการซื้อขายให้ผู้ใช้ทราบ โดยผ่านฟังก์ชันอินเทอร์เฟซที่มีอยู่ การทำงาน คือ ภายหลังจากการทำรายการเสนอซื้อขายหลักทรัพย์แล้วจึงส่งรายการนั้นไปยังตลาดหลักทรัพย์เพื่อประมวลผล และตลาดจะส่งผลการซื้อขายที่สำเร็จมาบันทึกใน

โฮสต์ของโบกเกอร์ให้โดยอัตโนมัติเมื่อมีการตกลงซื้อขายของรายการจากโบรกเกอร์รายนั้น ส่วนฟังก์ชันการแสดงราคาเสนอซื้อขายหลักทรัพย์ โบรกเกอร์จะแจ้งคำร้องขอไปยังตลาดหลักทรัพย์เพื่อให้ส่งข่าวสารที่จำเป็นกลับมา

1.3 ขั้นตอนการทำงาน

เริ่มจากการศึกษาหาข้อมูลเกี่ยวกับเรื่องการประมวลผลแบบกระจาย การพัฒนาแอพพลิเคชันโดยใช้รีโมตโพรซีเจอร์คอล (RPC) และ อินเตอร์โพรเซสคอมมิวนิเคชัน (IPC) พร้อมกันนั้นก็ทำการค้นคว้าและจัดหาข้อมูลที่เกี่ยวกับระบบการซื้อขายหลักทรัพย์ที่ใช้งานจริงในตลาดหลักทรัพย์แห่งประเทศไทย ทั้งโดยการค้นคว้าจากหนังสือ , การสัมภาษณ์จากผู้มีความรู้ในการซื้อขายหลักทรัพย์ และการไปศึกษาการซื้อขายในบริษัทหลักทรัพย์ ที่ทำงานเป็นโบรกเกอร์จริง เพื่อกำหนดขอบเขตฟังก์ชันการทำงานของระบบจำลอง และเรียนรู้อัลกอริทึมที่เหมาะสมกับปัญหาแม้อาจจะไม่ใช่อัลกอริทึมการซื้อขายจริงในระบบซื้อขายอัตโนมัติของตลาดหลักทรัพย์แห่งประเทศไทย

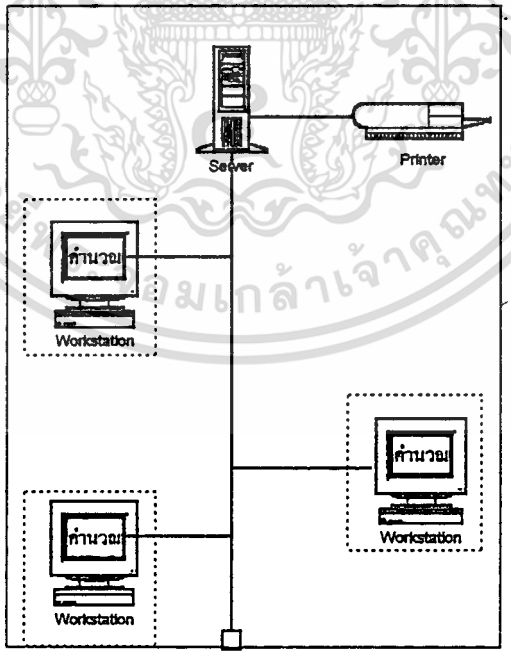
จากนั้นจึงออกแบบแอพพลิเคชัน โดยมองลักษณะฟังก์ชันการทำงานของโบรกเกอร์และตลาดหลักทรัพย์แยกออกจากกัน แต่ออกแบบให้มีความสัมพันธ์ซึ่งกันและกันอย่างเหมาะสม ทำการพัฒนาฟังก์ชันของส่วนโบรกเกอร์ หรือ ไคล์เอ็นต์ หรือ ผู้ขอบริการ แยกต่างหากจากฟังก์ชันของส่วนตลาดหลักทรัพย์ หรือ เซอร์ฟเวอร์ หรือ ผู้ให้บริการ จนสามารถทำงานในฟังก์ชันพื้นฐานของตนได้สมบูรณ์ แล้วจึงเชื่อมการทำงานของสองส่วนนี้เข้าด้วยกันอีกครั้ง และปรับปรุงให้ทำงานถูกต้องสัมพันธ์กันได้ในที่สุด

บทที่ 2
ทฤษฎีและหลักการ

2.1 ไคล์เอ็นต์เซอร์ฟเวอร์

ในหน่วยงานและสถานศึกษาในปัจจุบัน คงไม่มีใครปฏิเสธได้ว่าระบบเครือข่ายได้เข้ามามีบทบาทต่อชีวิตประจำวันของพนักงานและนักศึกษาเป็นอย่างมาก ระบบเครือข่ายได้เริ่มเข้ามาแทนที่ระบบใหญ่ ๆ อย่างเมนเฟรมและมินิคอมพิวเตอร์แล้ว สาเหตุที่มันสามารถเข้ามาแทนที่ได้ก็คือ การคำนวณแบบกระจายศูนย์หรือที่เรียกว่า distributed processing นั่นเอง ซึ่งสามารถแบ่งงานกันทำในแต่ละเวอร์กสเตชันได้ ในขณะที่เครื่องเมนเฟรมและมินิคอมพิวเตอร์ส่วนใหญ่ยังใช้การคำนวณแบบ central processing หรือระบบการคำนวณจากส่วนกลางอยู่

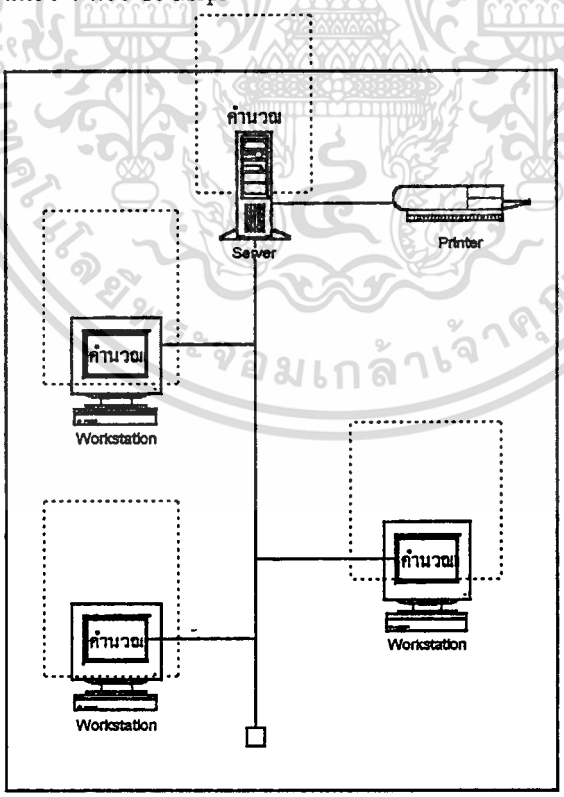
ปัจจุบันผู้ประกอบการในเครื่องระดับเมนเฟรมและมินิคอมพิวเตอร์ เริ่มหันมาวางโครงสร้างของตัวเองใหม่ให้มาเป็นรูปแบบการคำนวณแบบกระจายศูนย์กันแล้ว ซึ่งมีหลายแบบ แต่ละแบบมีประสิทธิภาพที่แตกต่างกัน แต่แบบที่กำลังนิยมกันอยู่ในขณะนี้ก็คือแบบที่เรียกว่า Client/Server processing ส่วนใหญ่เราจะพบการทำงานของระบบไคล์เอ็นต์เซอร์ฟเวอร์นี้ในระบบเครือข่ายแลน ระบบนี้ยังแบ่งออกได้เป็นสองชนิดคือ Shared-device processing และแบบ Client/Server processing



รูปที่ 2.1-1 การทำงานของ Share-device

ระบบแบบ Shared-device processing ถูกพัฒนาขึ้นมาก่อนเพื่อตอบสนองการขยายตัวอย่างรวดเร็วของระบบเครือข่ายแบน ระบบแบบนี้มีข้อดีและมีความรวดเร็ว ระบบนี้แสดงอยู่ใน รูปที่ 2.1-1 การทำงานของแอพพลิเคชันจะวิ่งอยู่บนโฮสต์ ซึ่งในระบบแบบนี้จะเรียกโฮสต์ว่า เซอร์ฟเวอร์ การทำงานส่วนใหญ่ของมันจะยอมให้ผู้ใช้มาร่วมใช้รีซอร์สต่าง ๆ ที่อยู่บนเซอร์ฟเวอร์ได้ แต่รีซอร์สจะถูกจำกัดให้เป็นเพียงการร่วมกันใช้ไฟล์และเครื่องพิมพ์เท่านั้น (พื้นที่ในกรอบสี่เหลี่ยมเส้นประ คือ พื้นที่ที่มีการคำนวณ)

ข้อเสียของมันก็คือ การทำงานส่วนใหญ่จะอยู่ในเครื่องเวอร์กสเตชันของผู้ใช้เท่านั้น ยกเว้นฟังก์ชันการพิมพ์และการทำไฟล์ I/O ที่จะอยู่บนเครื่องเซอร์ฟเวอร์ เมื่อผู้ใช้ต้องการใช้แอพพลิเคชันหรือไฟล์ มันจะทำการร้องขอมาที่เซอร์ฟเวอร์ เซอร์ฟเวอร์จะส่งไฟล์ทั้งหมดหรือ แอพพลิเคชันนั้น ๆ เข้ามาทำงานต่อไป เซอร์ฟเวอร์ที่ทำหน้าที่ในการแบ่งไฟล์กันใช้นี้เรียกว่า File server ฟังก์ชันทางการพิมพ์ก็คล้าย ๆ กัน คือ เมื่อผู้ใช้ต้องการพิมพ์ไฟล์ข้อมูล มันก็จะส่งไฟล์นั้นไปให้เซอร์ฟเวอร์เพื่อส่งออกมาพิมพ์ เซอร์ฟเวอร์ที่ทำหน้าที่นี้จะเรียกว่า Print server บางครั้งเซอร์ฟเวอร์ตัวเดียวสามารถทำหน้าที่ของ File server และ Print server ได้ ระบบเครือข่ายที่สามารถรองรับการทำงานแบบนี้ได้จะต้องเป็นระบบที่มีความเร็วพอสมควร เพราะข้อมูลที่วิ่งไปมาในระบบมีขนาดใหญ่ (ส่งรับข้อมูลทั้งไฟล์หรือดาต้าเบส) ระบบที่นิยมใช้ โดยทั่วไปคือ Novell Netware และ Microsoft LAN Manager เป็นต้น ทั้งสองตัวจะวิ่ง โดยมาตรฐานหลายอย่างรวมทั้ง Ethernet ที่มีความเร็ว 10 MBps หรือ Token Ring ที่มีความเร็ว 4 หรือ 16 Mbps



รูปที่ 2.1-2 การทำงานของ Client/Server

เซอร์ฟเวอร์ที่ใช้ในระบบไม่จำเป็นต้องมีขนาดซีพียูที่ใหญ่โตซึ่งมีราคาแพงอย่างในระบบเมนเฟรมและมินิคอมพิวเตอร์อีกต่อไปแล้ว แต่เครื่องเวอร์กสเตชันจำเป็นจะต้องมีซีพียูที่สามารถวิ่งแอพพลิเคชันของตนเองได้ ความเร็วของการทำงานก็จะถูกจำกัดอยู่ในเครื่องเวอร์กสเตชันนั่นเอง ระบบ Client/Server processing จะคล้ายกับระบบ Shared-device processing มาก แต่แทนที่การคำนวณทางด้านแอพพลิเคชันจะอยู่ในเครื่องเวอร์กสเตชันของผู้ใช้ทั้งหมด มันจะแบ่งการคำนวณมาอยู่ในตัวเซอร์ฟเวอร์ด้วย ดังแสดงไว้ในรูปที่ 2.1-2 (พื้นที่ในกรอบสี่เหลี่ยมเส้นประ คือ พื้นที่ที่มีการคำนวณ)

จะเห็นว่าแอพพลิเคชันทำงานทั้งในเครื่องเซอร์ฟเวอร์และเครื่องเวอร์กสเตชันด้วย เมื่อผู้ใช้ต้องการข้อมูลบางส่วนที่อยู่ในดาต้าเบสไฟล์ขนาดใหญ่ เครื่องเวอร์กสเตชันจะส่งคำร้องขอข้อมูลนั้นไปที่ตัวเซอร์ฟเวอร์ เซอร์ฟเวอร์จะทำการคำนวณและดึงข้อมูลเหล่านั้นออกมา และส่งเฉพาะข้อมูลนั้น ๆ กลับมายังเครื่องเวอร์กสเตชันที่ร้องขอมา เมื่อเวอร์กสเตชันได้รับข้อมูลมันก็จะนำมาคำนวณต่อไป

จะเห็นได้ว่าข้อมูลที่ส่งไปมาในระบบจะมีเพียงข้อมูลที่จำเป็นเท่านั้น ในขณะที่ถ้าส่งในระบบ Shared-device processing จะเป็นการส่งไฟล์ดาต้าเบสใหญ่ ๆ ทั้งไฟล์มาที่เครื่องเวอร์กสเตชัน ทั้ง ๆ ที่มันต้องการข้อมูลเพียงไม่กี่ไบต์ก็ตาม เมื่อสถาบันหรือหน่วยงานใดมีการใช้แอพพลิเคชันแบบ Shared-device บนระบบเครือข่ายมากเกินไป จะทำให้ระบบเครือข่ายนั้น ๆ ไม่สามารถรองรับ traffic ที่มากเช่นนั้นได้ เวลาในการตอบสนองของระบบต่อผู้ใช้ก็จะมีค่าสูงขึ้นด้วย แต่ถ้าหน่วยงานใดหันมาใช้แอพพลิเคชันแบบไคล์เอ็นต์เซอร์ฟเวอร์มากขึ้น คุณก็ไม่จำเป็นต้องมากังวลถึง traffic ที่จะเกิดขึ้นในระบบเครือข่ายของคุณเลย เพราะระบบไคล์เอ็นต์เซอร์ฟเวอร์จะใช้ traffic น้อยมากเมื่อเทียบกับระบบแบบ Shared-device

ด้วยการส่งข้อมูลที่มีประสิทธิภาพแบบนี้ ทำให้การติดต่อกันข้าม WAN นี้สามารถทำได้รวดเร็วมาก ความสามารถและประสิทธิภาพในระบบก็มีสูงขึ้นด้วย อีกทั้งแอพพลิเคชันที่วิ่งอยู่บนตัวเซอร์ฟเวอร์ที่คอยให้บริการทางด้านข้อมูลแก่ผู้ใช้หรือไคล์เอ็นต์ก็ไม่จำเป็นจะต้องเป็นแอพพลิเคชันแบบเดียวกัน แต่ขอให้มีมาตรฐานแบบเดียวกันก็พอ ตัวอย่างของแอพพลิเคชันที่วิ่งอยู่ในเซอร์ฟเวอร์ก็คือ ดาต้าเบสซอฟต์แวร์อย่าง Sybase และ Microsoft SQL Server เป็นต้น แอพพลิเคชันทางดาต้าเบสเหล่านี้จะอยู่บน NOS (Network Operating System) อย่าง Netware และ MS LAN Manager ที่เวอร์กสเตชันของไคล์เอ็นต์ ผู้ใช้สามารถใช้แอพพลิเคชันอย่าง MS Access, Excel และ Paradox เป็นต้น เพื่อทำหน้าที่ร้องขอบริการจากเซอร์เวอร์

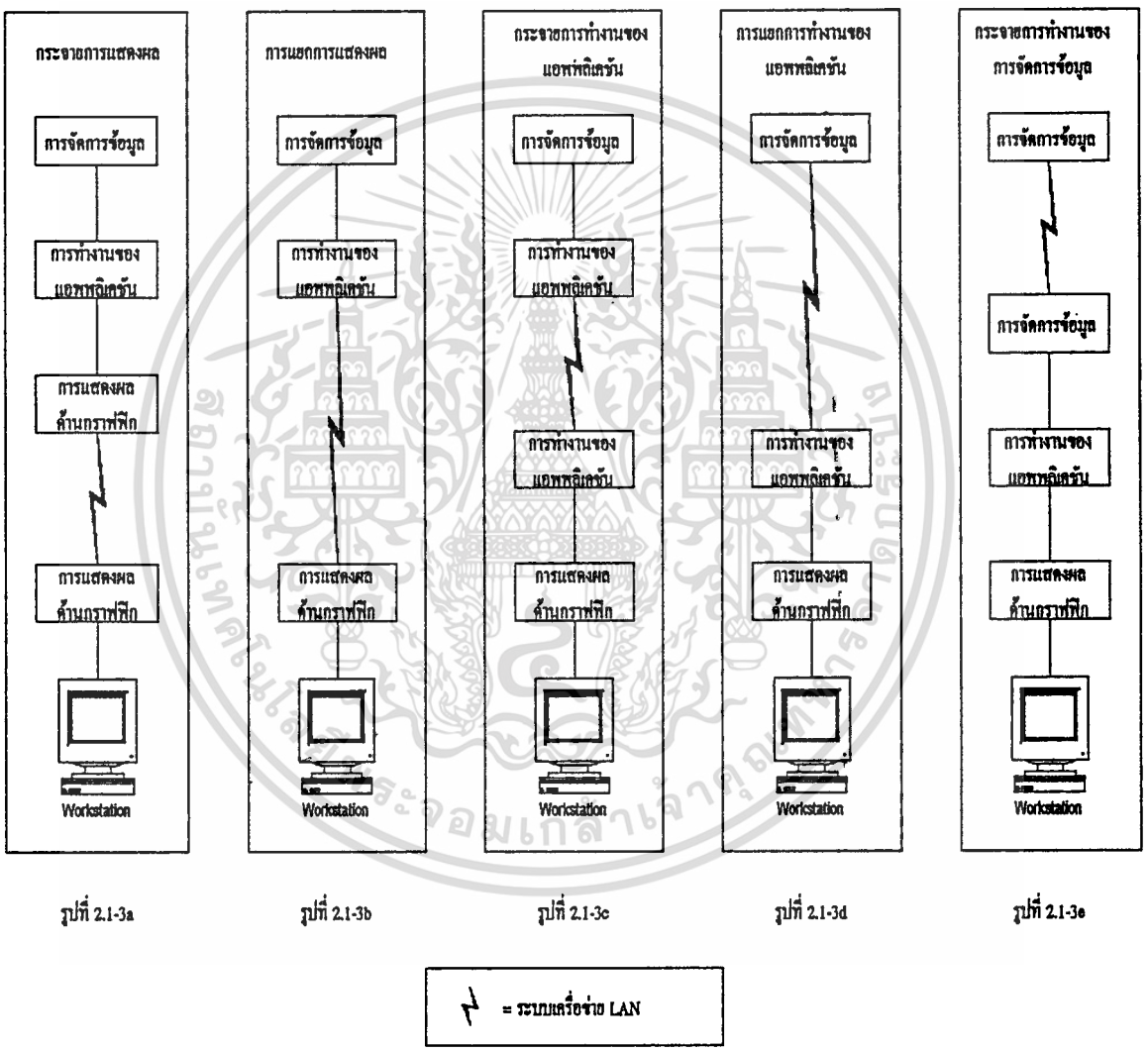
ระบบไคล์เอ็นต์เซอร์ฟเวอร์ยังสามารถแบ่งอย่างละเอียดได้เป็น 5 ชนิด คือ

2.1.1 ชนิดการกระจายการแสดงผล (Distributed Presentation)~

เป็นการทำงานภายใต้ Graphical User Interface หรือ GUI อย่างในวินโดวส์ เซอร์ฟเวอร์จะทำการคำนวณ และส่งภาพกราฟิกที่ใช้ติดต่อกับผู้ใช้มาที่เครื่องเวอร์กสเตชันเมื่อมันร้องขอมา ในขณะเดียวกันผู้ใช้ก็จะไม่สามารถบอกความแตกต่างได้เลยว่า ภาพที่ปรากฏอยู่นั้นเป็นภาพที่มาจากเซอร์ฟเวอร์หรือจากในตัวเครื่องเอง การทำงานของระบบไคล์เอ็นต์เซอร์ฟเวอร์ชนิดนี้จะเป็นแบบที่กระจายการทำงานน้อยที่สุดของทั้งห้าชนิด (รูปที่ 2.1-3a)

~2.1.2 ชนิดแยกการแสดงผล (Remote Presentation)~

จากรูปที่ 2.1-3b จะเห็นว่าการแสดงผลต่าง ๆ ต่อผู้ใช้จะถูกทำจากเวอร์กสเตชันหรือไคล์เอ็นต์เท่านั้น การทำงานของแอพพลิเคชันลอจิกจะอยู่ในเซอร์ฟเวอร์ทั้งหมด รวมทั้งการจัดการข้อมูลด้วย ข้อมูลที่ส่งข้ามระบบเครือข่ายก็จะมีน้อยกว่าชนิดที่หนึ่ง เพราะว่ามันไม่ต้องส่งภาพกราฟิกที่มีขนาดใหญ่ข้ามระบบไปให้ผู้ใช้เลย



รูปที่ 2.1-3 ชนิดต่าง ๆ ของระบบแบบไคล์เอ็นต์เซอร์ฟเวอร์

2.1.3 ชนิดการกระจายการทำงานของแอพพลิเคชัน (Distributed Logic)~

ระบบชนิดนี้จะมีการกระจายการทำงานของตัวแอพพลิเคชันเองให้อยู่ในทั้งตัวเซอร์ฟเวอร์และเวอร์กสเตชันดังเช่นในรูปที่ 2.1-3c จะเห็นได้ว่าเวอร์กสเตชันจะช่วยเซอร์ฟเวอร์ทำงานทางด้านแอพพลิเคชันเป็นส่วน

หน้า (Front-End) ซึ่งมีการเชื่อมโยงกับแอพพลิเคชันส่วนหลัง (Back-End) ที่อยู่ในตัวเซอร์ฟเวอร์ด้วย การคำนวณทางด้านแอพพลิเคชันก็จะถูกแบ่งออกจากเซอร์ฟเวอร์มากขึ้น ตัวอย่างเช่น ในเวอร์กสเตชันมีการคำนวณหลายชุด มันก็อาจส่งการคำนวณบางส่วนหรือทั้งหมดไปให้เซอร์ฟเวอร์คำนวณและจะให้ส่งผลลัพธ์กลับมาให้มัน

2.1.4 ชนิดแยกการทำงานของแอพพลิเคชัน (Remote Data Management)~

การทำงานของแอพพลิเคชันทั้งหมดจะอยู่ในตัวเวอร์กสเตชันทั้งหมด และเซอร์ฟเวอร์จะมีส่วนการจัดการข้อมูลเท่านั้น เมื่อเวอร์กสเตชันต้องการข้อมูลส่วนใดก็สามารถทำการร้องขอไปที่เซอร์ฟเวอร์ได้ เซอร์ฟเวอร์จะทำการค้นหาข้อมูลให้และส่งกลับมาที่เครื่องเวอร์กสเตชันดังรูปที่ 2.1-3d

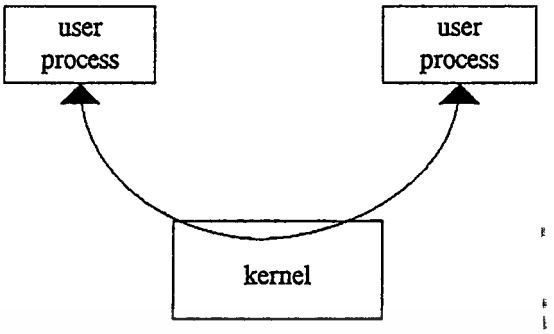
2.1.5 ชนิดกระจายการจัดการของข้อมูล (Distributed Database)~

เป็นชนิดที่นิยมใช้กันอย่างแพร่หลายในแอพพลิเคชัน Sybase หรือ Microsoft SQL Server เป็นต้น การคำนวณทางด้านแอพพลิเคชันทั้งหมดและบางส่วนของการจัดการข้อมูลจะอยู่ในเครื่องเวอร์กสเตชันและส่วนหลังของการจัดการข้อมูลจะอยู่ในเซอร์ฟเวอร์ดังรูป 2.1-3e วิธีนี้ สามารถเพิ่มประสิทธิภาพตัวเซอร์ฟเวอร์มากขึ้น เพราะการคำนวณส่วนใหญ่จะอยู่ในตัวเวอร์กสเตชัน แต่การค้นหาข้อมูลที่มีความซับซ้อนจะอยู่ในตัวเซอร์ฟเวอร์ แต่เวอร์กสเตชันอาจต้องมีความสามารถในการทำงานสูงพอควร การทำงานของทั้งระบบก็จะมีความรวดเร็วมาก~

2.2 Interprocess Communication (IPC)

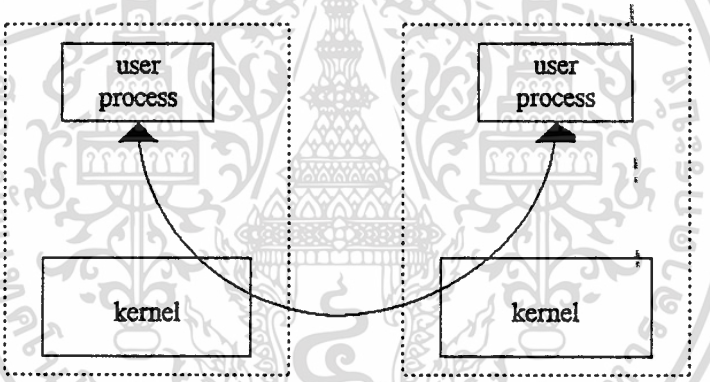
เนื่องจากการเขียนโปรแกรมบนเน็ตเวอร์กต้องเกี่ยวข้องกับการติดต่อของสองโพรเซส หรือ มากกว่าสองโพรเซส เราจึงต้องคำนึงถึงวิธีการติดต่อระหว่างโพรเซสที่ต่างกันอย่างระมัดระวัง ในการเขียนโปรแกรมหนึ่งโพรเซส การติดต่อระหว่างโมดูลอาจทำโดยการใช้ตัวแปรโกรบอล (grobal variable) แต่สำหรับโพรเซสที่แยกออกจากกันซึ่งต่างทำงานบน address space ของตัวเอง เราจำเป็นต้องมีรายละเอียดที่ต้องคำนึงมากขึ้น เมื่อสองโพรเซสต้องการติดต่อซึ่งกันและกัน ระบบปฏิบัติการจำเป็นต้องให้ความสะดวกสำหรับ Interprocess Communication (IPC) นี้

เราสามารถแสดงแผนภาพกรณีที่สองโพรเซสบนระบบคอมพิวเตอร์เดียวใช้ IPC เพื่อติดต่อกันได้ดังภาพ 2.2-1 ภาพนี้แสดงให้เห็นว่าข่าวสาร (information) ระหว่างสองโพรเซสเดินทางผ่านเคอร์เนล (kernel) ไปมาซึ่งกันและกัน



รูป 2.2-1 IPC ระหว่างสองโพรเซสบนหนึ่งระบบ

เมื่อขยายการติดต่อนี้เป็นโพรเซสบนระบบที่แยกจากกัน โดยใช้การติดต่อในรูปของเน็ตเวิร์กระหว่างสองระบบ เราสามารถแสดงเหตุการณ์ได้ดังภาพที่ 2.2-2



รูป 2.2-2 IPC ระหว่างสองโพรเซสบนต่างระบบ

วิธีของ IPC มีหลายวิธี ได้แก่

- pipes
- FIFOs (named pipes)
- message queues
- semaphores
- shared memory

สำหรับ message queues, semaphores, shared memory เป็นชุดของ System V IPC ที่ใช้งานด้วย system call

2.2.1 Message queue

แต่เฉพาะในโครงงาน ฯ ที่พัฒนานี้จะใช้ message queues เป็นหลัก คำสั่งที่เป็น system call ของ message queues แสดงดังตารางในตาราง 2.2-1

	Message queue
include file	<sys/msg.h>
system call to create or open	msgget
system call for control operations	msgctl
system calls for IPC operations	msgsnd
	msgrcv

ตาราง 2.2-1 system call ของ Message queues

System V จัดการกับ message โดยให้ message ทั้งหมดอยู่ใน kernel และจัดการโดย message queue identifier หรือ mewqid เพื่อใช้ชี้ไปยัง queue ของ message ที่กำลังทำงาน

แต่ละ message ใน queue จะต้องมีแอตทริบิวต์ดังนี้

- *type* ชนิด long integer
- *length* ความยาวของข้อมูลใน message (สามารถเป็นศูนย์)
- *data* ข้อมูล (ถ้า *length* มีค่ามากกว่าศูนย์)

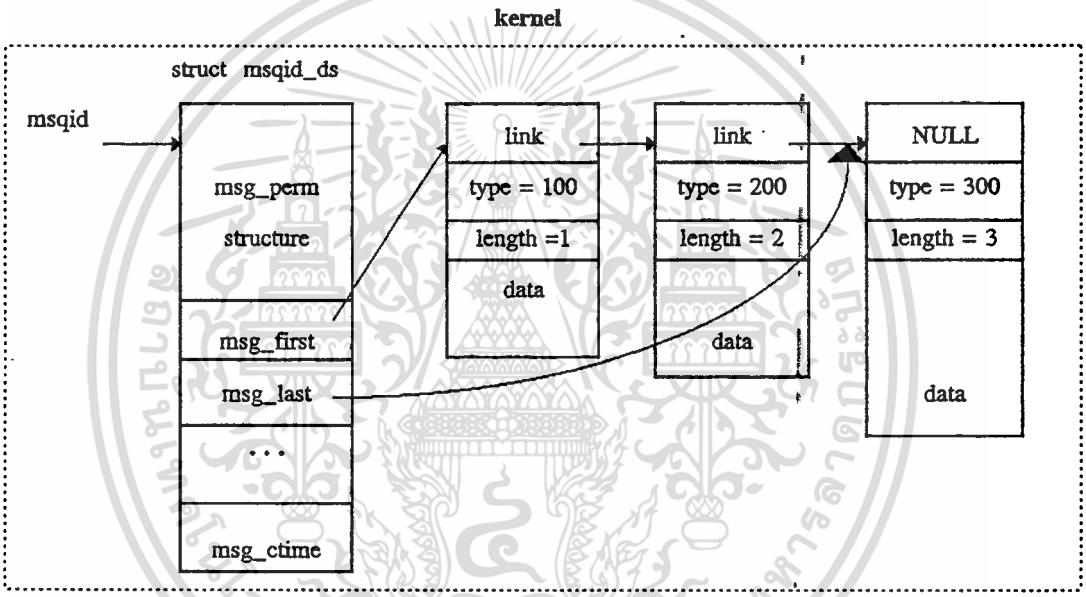
kernel จะดูแลรักษาทุก message queue ในระบบซึ่งทุก message queue จะประกอบด้วยข้อมูลตามโครงสร้างต่อไปนี้

```
#include <sys/types.h>
#include <sys/ipc.h>        /*defines the ipc_perm structure      */

struct  msqid_ds  {
        struct  ipc_perm    msg_perm;       /* operation permission struct      */
        struct  msg         *msg_first;     /* ptr to first message on q        */
        struct  msg         *msg_last;      /* ptr to last message on q         */
        ushort              msg_cbytes;     /* current # bytes on q      */
        ushort              msg_qnum;       /* current # of messages on q       */
        ushort              msg_qbytes;     /* max # of bytes allowed on q      */
        ushort              msg_lspid;      /* pid of last msgsnd       */
```

```
    ushort          msg_lrpid;      /* pid of last msgrcv      , */
    time_t          msg_stime;      /* time of last msgsnd       */
    time_t          msg_rtime;      /* time of last msgrcv       */
    time_t          msg_ctime;      /* time of last msgctl
                                       (that changed the above) */
};
```

เราสามารถแสดงภาพการทำงานของ message queue ใน kernel ที่เป็นลิงค์ลิสต์ (link list) ของ message ได้ดังรูป 2.2-3 สมมติให้มีสาม message ใน queue ซึ่งมีความยาว 1 ไบท์, 2 ไบท์และ 3 ไบท์ตามลำดับและสาม message นี้เขียนให้มี type คือ 100, 200 และ 300 ตามลำดับ



รูป 2.2-3 โครงสร้างเมสเสจคิว ในเคอร์เนล

2.2.2 การใช้คำสั่งของ Message queue

ก่อนใช้คำสั่งของเมสเสจคิว จะต้องประกาศอินคลูดไฟล์ ดังต่อไปนี้

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

คำสั่ง msgctl

```
msgctl(id, cmd, buf)
int id, cmd;
struct msqid_ds *buf;
```

msgctl อนุญาตให้โพรเซสเซ็ทหรืออ่านค่าสถานะของ message queue id หรือ ยกเลิกคิว ขึ้นอยู่กับค่าของ cmd ที่ผู้ใช้กำหนดให้ โครงสร้างของ msqid_ds ประกอบด้วย

```
struct ipc_perm {
        ushort    uid     /* user id ปัจจุบัน */
        ushort    gid;    /* group id ปัจจุบัน */
        ushort    cuid;   /* user id ของผู้สร้าง */
        ushort    cgid;   /* group id ของผู้สร้าง */
        ushort    mode;   /* access modes */
        short     pad1;   /* ใช้โดยระบบ */
        long      pad2;   /* ใช้โดยระบบ */
};

struct msqid_ds {
        struct ipc_perm   msg_perm;     /* permission struct */
        short             pad1[7];      /* ใช้โดยระบบ */
        ushort            msg_qnum;     /* จำนวนของเมสเสจในคิว */
        ushort            msg_qbytes;   /* จำนวนไบท์มากที่สุดบนคิว */
        ushort            msg_lspid;    /* pid ของคำสั่ง msgsnd สุดท้าย */
        ushort            msg_lrpid;    /* pid ของคำสั่ง msgrcv สุดท้าย */
        time_t            msg_stime;    /* เวลาใช้ msgsnd ครั้งสุดท้าย */
        time_t            msg_rtime;    /* เวลาใช้ msgrcv ครั้งสุดท้าย */
        time_t            msg_ctime;    /* เวลาที่เปลี่ยนแปลงครั้งสุดท้าย */
};
```

คำสั่งและความหมายของคำสั่งเมื่อกำหนดค่า cmd ให้ มีดังนี้

IPC_STAT	อ่านเฮดเดอร์ของเมสเสจคิว ตาม id ใน buf
IPC_SET	ตั้งค่าของ msg_perm.uid, msg_perm.gid, msg_perm.mode (9 บิทต่ำ) และ msg_qbytes จากค่าใน buf
IPC_RMID	ยกเลิกเมสเสจคิว id

คำสั่ง msgget

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

msgget รีเทิร์นค่าชี้ไปยังเมสเสจคิว คือ id ของคิวที่ชื่อ key ซึ่งสามารถระบุให้คิว key นั้นรีเทิร์นค่า id ที่เป็น private queue (IPC_PRIVATE) เมื่อเป็นคิวใหม่ที่สร้าง, สามารถระบุ Flag ให้เป็นคิวที่ถูกสร้าง (IPC_CREAT) และสามารถทำให้การสร้างเป็นแบบ exclusive (IPC_EXCL) ซึ่ง msgget จะเฟลถ้ามีคิวอยู่แล้ว

คำสั่ง msgsnd และ msgrcv

```
msgsnd(id, msgp, size, flag)
int id, size, flag;
struct msgbuf *msgp;

msgrcv(id, msgp, size, type, flag)
int id, size, type, flag;
struct msgbuf *msgmp;
```

msgsnd จะส่งเมสเสจที่มีไบท์เท่ากับ size ในบัฟเฟอร์ msgp ไปยังเมสเสจคิว id และ msgbuf อยู่ในรูปแบบคือ

```
struct msgbuf {
        long mtype;
        char mtext[ ];
};
```

ถ้าบิท IPC_NOWAIT ถูกเซ็ตเป็น off ใน flag แล้ว msgsnd จะหลับถ้าจำนวนไบท์บนเมสเสจคิวมากกว่าจำนวนมากที่สุด หรือถ้าจำนวนของเมสเสจของระบบเกินกว่าค่าที่มากที่สุด แต่ถ้า IPC_NOWAIT ถูกเซ็ต msgsnd จะรีเทิร์นทันทีในกรณีเดียวกัน

msgrcv ทำหน้าที่รับเมสเสจจากคิวที่ระบุโดย id ถ้า type เป็น 0 ก็จะรับเมสเสจแรกในคิว, ถ้าเป็นค่าบวก เมสเสจแรกที่มีชนิดเป็น type จะถูกรับมาก่อน, และถ้าเป็นลบ จะรับเมสเสจแรกที่มีค่า type ต่ำสุดที่น้อยกว่าหรือเท่ากับ type ค่า size จะบอกค่าขนาดที่ใหญ่ที่สุดของข้อความเมสเสจที่ผู้ใช้ต้องการจะรับ ถ้า MSG_NOERROR ถูกเซ็ทใน flag เคอร์เนลจะ truncate เมสเสจที่รับถ้ามีขนาดใหญ่กว่า size มิฉะนั้นมันจะรีเทิร์น error ถ้า IPC_NOWAIT ไม่ถูกเซ็ทใน flag แล้ว msgrcv จะหลับจนกว่าเมสเสจที่มี type ที่ต้องการจะถูกส่งมา แต่ถ้า IPC_NOWAIT ถูกเซ็ต มันจะรีเทิร์นทันที msgrcv จะรีเทิร์นค่าจำนวนไบท์ในข้อความเมสเสจ

ตัวอย่าง โปรแกรมของไคล์เอ็นท์ และ โปรแกรมของเซิร์ฟเวอร์ที่ใช้คำสั่งของเมสเสจคิว มีดังต่อไปนี้

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

#define MSGKEY      75

struct msgform {
        long    mtype;
        char    mtext[256];
};

main()
{       struct msgform msg;
        int msgid, pid, *pint;

        msgid = msgget(MSGKEY, 0777);
        pid = getpid();
        pint = (int *) msg.mtext;
        *pint = pid;        /* copy pid into message text */
        msg.mtype = 1;

        msgsnd(msgid, &msg, sizeof(int), 0);
        msgrcv(msgid, &msg, 256, pid, 0);          /* pid is used as the msg type */
        printf("client: receive from pid %d\n", *pint);
}
```

รูป 2.2-5 โพรเซสของไคล์เอ็นต์

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

#define MSGKEY      75
struct msgform
{
        long mtype;
        char mtext[256];
} msg;
int msgid;

main()
{
        int I, pid, *pint;
        extern cleanup();

        for(i = 0; i < 20; I++)
                signal(i, cleanup);
        msgid = msgget(MSGKEY, 0777 | IPC_CREAT);

        for (;;)
        {
                msgrcv(msgid, &msg, 256, 1, 0);
                pint = (int *) msg.mtext;
                pid = *pint;
                printf("server: receive from pid %d\n", pid);
                msg.mtype = pid;
                *pint = getpid();
                msgsnd(msgid, &msg, sizeof(int), 0);
        }
}

cleanup()
{
        msgctl(msgid, IPC_RMID, 0);
        exit();
}
```

รูป 2.2-6 โพรเซสของเซอร์ฟเวอร์

2.2.3 ข้อจำกัดของ Message Queue

การใช้เมสเสจคิวย่อมมีค่าจำกัดในแต่ละระบบ ซึ่งอาจเปลี่ยนแปลงได้โดยแอดมินนิสเตเตอร์ (administrator) โดยกำหนดค่าคอนฟิกในเคอร์เนลใหม่ ตาราง 2.2-2 แสดงค่าจำกัดสำหรับเวอร์ชันที่แตกต่างกันของยูนิกซ์ทั้ง 5 เวอร์ชัน

Parameter name	AT&T VAX Sys V Rel 2	DEC VAX Ultrix	AT&T Unix PC	Xenix 286	Xenix 386	Description
msgmax	8,192	8,192	8,192	8,192	8,192	max # of bytes per message
msgmnb	16,384	16,384	16,384	8,192	8,192	max # of bytes on any one message queue
msgni	50	50	50	10	10	max # of message queues,systemwide
msgtql	40	40	40	60	60	max # of messages, system wide
	8,192	8,192	8,192	8,192	8,192	max # of bytes of messages, systemwide

ตาราง 2.2-2 ค่าจำกัดของเมสเสจคิว

ค่าต่าง ๆ ที่แสดงนี้ช่วยสำหรับการวางแผนการเขียนโปรแกรมให้สามารถเข้าได้กับระบบ เป็นต้นว่า ถ้าเราเขียนแอพพลิเคชันที่ใช้เมสเสจที่มีความยาว 20,000 ไบท์ จะไม่สามารถทำงานได้ ค่า 8192 ไบท์สำหรับขนาดเมสเสจที่ใหญ่ที่สุดนี้ก็คล้ายคลึงกับค่าขนาดข้อมูลที่ใหญ่ที่สุดที่เขียนลงไปป์ (pipe) แบบ FIFO คือ 4096 ไบท์นั่นเอง

2.3 Remote Procedure Calls (RPC)

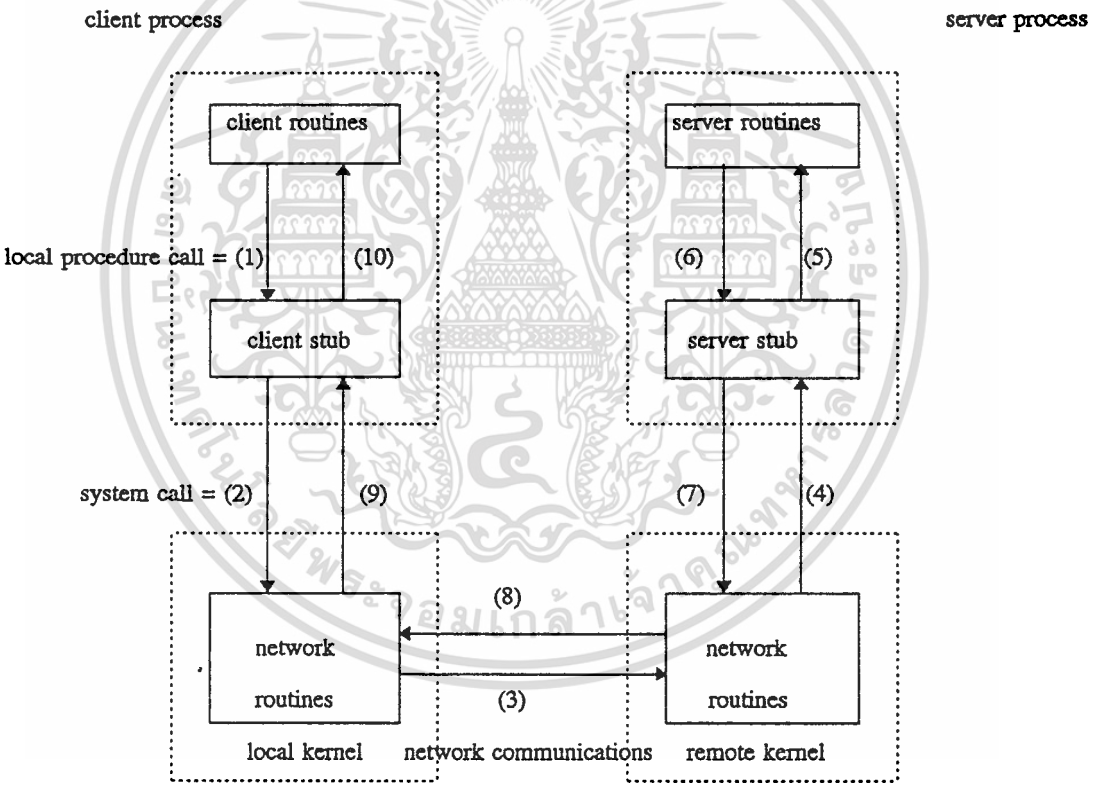
2.3.1 RPC คืออะไร

การเรียกโพรซีเดอร์ (procedure call) (บางครั้งหมายถึงการเรียกฟังก์ชัน หรือการเรียกรูทีนย่อย) เป็นวิธีการที่รู้จักกันดีว่าเป็นการย้ายการควบคุมจากส่วนหนึ่งของโพรเซสไปยังอีกส่วนหนึ่ง และจะส่งคืน (return) การควบคุมกลับมายังผู้เรียก สถาปัตยกรรมคอมพิวเตอร์หลายแบบกำหนดการทำงานนี้ด้วยการใช้คำสั่งในรูปของ "jump-to-subroutine" การเรียกโพรซีเดอร์จะมีการส่งผ่านค่าอาร์กิวเมนต์ (argument) จากผู้เรียก (client) ไปยัง

ผู้ถูกเรียก (server) และมักจะส่งคืนหนึ่งหรือมากกว่าหนึ่งค่าจากโพรซีเดอร์ที่ถูกเรียก ระบบส่วนใหญ่ในปัจจุบันจะมีผู้เรียกและผู้ถูกเรียกอยู่ในหนึ่งโพรเซสในโฮส (host) นั้น มันจะถูกเชื่อมกันโดยใช้ลิงค์เอดดิเตอร์ (link editor) เมื่อไฟล์ของโปรแกรมถูกสร้าง เราจะเรียกการเรียกโพรซีเดอร์ชนิดนี้ว่า local procedure call

ส่วน remote procedure call (RPC) จะมีลักษณะคือ โพรเซสบน local system เรียกโพรซีเดอร์บน remote system เหตุผลที่เราเรียกการ call เช่นนี้ว่า "procedure call" ก็เพราะว่า การทำงานที่ผู้เขียนโปรแกรมรู้สึกจะเหมือนกับการเรียกโพรซีเดอร์ธรรมดา โดยที่ผู้เขียนโปรแกรมไม่ต้องจัดการการติดต่อระหว่างชั้นต่าง ๆ ของโปรโตคอลเอง

รูป 2.3-1 จะแสดงขั้นตอนการติดต่อแบบ Remote Procedure Call และหมายเลขที่กำกับขั้นตอนต่าง ๆ ในภาพมีอธิบายดังนี้



รูป 2.3-1 แสดงผังการติดต่อทางไกล (remote procedure call model)

(1) ผู้ขอบริการจะเรียกใช้ฟังก์ชั่นที่ให้บริการจากฝั่งรีโมท แบบโลคอลจากส่วน client stub ซึ่งมันจะคิดว่าส่วน client stub นี้เป็นโพรซีเดอร์ของผู้ให้บริการที่มันต้องการเรียกใช้ จุดประสงค์ของ stub ก็คือมันจะทำการรวบรวมค่าส่งผ่านต่างๆไปยังรีโมทโปรซีเยอร์ในรูปของมาตราฐานหนึ่ง และจะทำการสร้างให้เป็นข้อมูลเครือข่าย (network messages) ส่งผ่านไปในเนตเวอร์คเรียกว่า marshaling

(2) ข้อมูลเครือข่ายนี้จะถูกส่งไปยังระบบรีโมทโดย client stub มันจะรเรียกใช้ส่วน local kernel แบบ system call

(3) ข้อมูลเครือข่ายจะถูกส่งไปยังฝั่งของ remote system โดยผ่านทางเส้นทางการเชื่อมต่อ

(4) โปรแกรมที่เป็น server stub ของฝั่งรีโมทจะคอยคำขอจากผู้ขอบริการ server stub จะเปลี่ยนแปลงรูปแบบของข้อมูลที่ส่งมาให้อยู่ในรูปของการเรียกฟังก์ชันแบบโลคอลธรรมดา

(5) ทำการส่งคำสั่งเรียกใช้ฟังก์ชั่นไปยังส่วนโปรแกรมของผู้ให้บริการ เพื่อทำการประมวลผลตามคำสั่งนั้น

(6) เมื่อผู้ให้บริการประมวลผลเสร็จแล้ว ก็จะส่งผลกลับมายัง server stub

(7) server stub จะรวบรวมผลและสร้างเป็นข้อมูลเครือข่ายอีกครั้งหนึ่งเพื่อส่งต่อไปยัง client stub

(8) ข้อมูลจะถูกส่งกลับผ่านเส้นทางเครือข่ายไปยัง client stub

(9) client stub จะอ่านข้อมูลจากส่วน local kernel

(10) และปรับเปลี่ยนข้อมูลที่ได้รับมาให้อยู่ในรูปผลการเรียกใช้ฟังก์ชันแบบธรรมดา ส่งกลับไปยังส่วนผู้เรียกใช้บริการ

หลักการของ remote procedure call ก็คือมันจะทำการซ่อนรหัสการติดต่อในเครือข่ายไว้ใน stub ซึ่งจะช่วยลดความยากในการเขียนโปรแกรมแบบ client/server เกี่ยวกับรายละเอียดในการติดต่อ (socket) ซึ่งเป็นผลดีที่ช่วยในการพัฒนาโปรแกรมประยุกต์แบบกระจายได้ง่ายขึ้น RPC จะอยู่ในชั้นการติดต่อระหว่างชั้น transport layer ถึงชั้น application layer ในมาตรฐานการเชื่อมต่อแบบ OSI

RPC ที่ใช้พัฒนาระบบจำลองตลาดหลักทรัพย์ในโครงงานนี้ เป็นของ Sun Microsystems' Open Network Computing (ONC) ONC ประกอบด้วยสองส่วนคือ RPC (the Remote Procedure Call specification) และ XDR (the eXternal Data Representation standard) ระบบ Sun RPC นี้สามารถใช้โปรโตคอลการสื่อสารทั้ง UDP และ TCP

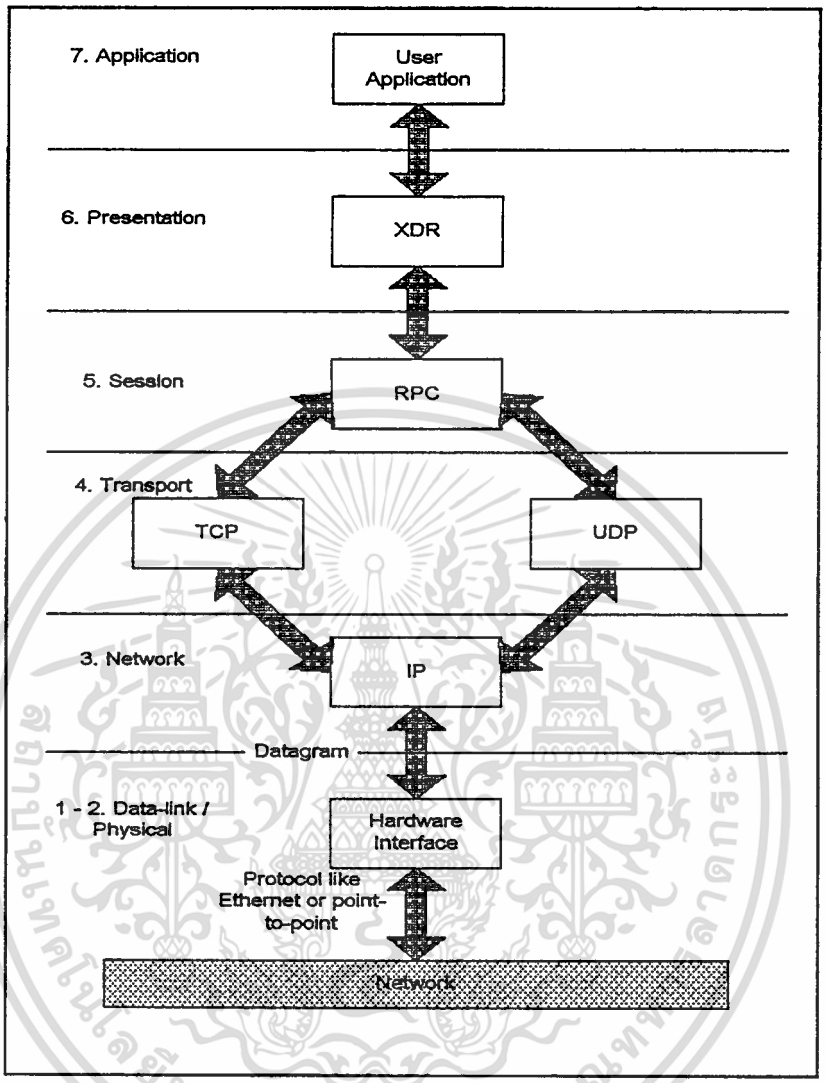
2.3.2 การพัฒนา Application โดยใช้ RPC

การพัฒนาโปรแกรมที่ใช้ RPC จะประกอบด้วยขั้นตอนดังต่อไปนี้ คือ

- กำหนด โปรโตคอล ที่จะใช้ในการติดต่อระหว่างผู้เรียก และ ผู้ให้บริการ
- เขียน code ส่วนของผู้เรียก และผู้ให้บริการ
- จากนั้นจะเป็นการ compile แล้ว link เข้ากับ stub และ libraries แล้วจึงทำการทดสอบโดยการ run โปรเซสผู้เรียกบนอีกเครื่องหนึ่งบนเครือข่าย เพื่อตรวจสอบผลการทำงาน

2.3.2.1 การกำหนด RPC โปรโตคอล

ขั้นแรกของการสร้างโปรแกรมด้วยRPC จะต้องทำการกำหนดชื่อของ procedures และชนิดของ parameter และ return argument



รูป 2.3-2 ตำแหน่งของ RPC และ XDR ใน OSI Model

ที่จะอนุญาตให้ผู้เรียกสามารถเรียกใช้ได้ซึ่งโปรโตคอลคอมไพเลอร์จะใช้ข้อกำหนดโปรโตคอล(Protocal Definitions) ที่เรากำหนดไว้นี้ในการสร้าง stub ทั้งส่วนของผู้เรียก และ ส่วนของผู้ให้บริการ รวมทั้งส่วน XDR filter ออกมาให้ ภาษาที่ใช้ในการกำหนดโปรโตคอล ของ RPC นั้นเรียกว่า ONC RPC Language(RPCL) ซึ่งมีไวยากรณ์บางส่วนคล้ายกับภาษา C และ Pascal โดยที่ประกอบไปด้วยข้อกำหนดต่างๆซึ่งได้แก่

- Constant (const)
- Enumerations (enum)
- Structure (struct)
- Union (union)
- Typedefinition (typedef)

- Program (program)

enum,struct และ typedef นั้นจะเหมือนกับที่ใช้ในภาษา C และ const นั้นใช้ในการกำหนดชื่อ(lable) ให้กับค่าคงที่แบบจำนวนเต็ม ส่วน union ของ RPCL นั้นจะเหมือนกับ variant record ในภาษา Pascal และ program จะเป็นส่วนที่ใช้ในการประกาศ procedure ต่างๆที่จะอนุญาตให้เรียกใช้ได้ ซึ่งไฟล์ที่ใช้เก็บข้อกำหนดโปรโตคอลนี้ จะมีนามสกุลต่อท้ายด้วย ".x" ซึ่งตัวอย่างของข้อกำหนด ได้แสดงไว้ในรูปที่ 2.3-3

```
/* preprocessor directives */
    %#define DATABASE "personnel.dat"

    /* constant definitions */
    const MAX_STR     = 256;

    /* structure definitions, no enumerations needed */
    struct record  {
        string    firstName<MAX_STR>;
    /* < > defines the maximum possible length */
        string    middleInitial<MAX_STR>;
        string    lastName<MAX_STR>;
        int       phone;
        string    location<MAX_STR>;
};
 /* program definition, no union or typedef definitions needed*/      program
RDBPROG { /* could manage multiple servers */
          version RDBVERS  {
                record FIRSTNAME_KEY(string)   = 1;
                record LASTNAME_KEY(string)    = 2;

                record PHONE_KEY(int)          = 3;
                record LOCATION_KEY(string)    = 4;
                int    ADD_RECORD(record)      = 5;
          } = 1;  /* set the interface version number */
}    = 0x20000001; /* program number ranges established by ONC */
```

รูปที่ 2.3-3 แสดงตัวอย่างข้อกำหนดโปรโตคอล RPCL: rdb.x

ซึ่งจะเห็นว่าโปรซีเยอร์ของผู้ให้บริการในการค้นหาข้อมูลและเพิ่ม record ของฐานข้อมูล มีทั้งหมดห้าโปรซีเยอร์ด้วยกัน โดยแต่ละโปรซีเยอร์จะมีหมKยเลขประจำตัวซึ่งเริ่มจาก 1 เป็นต้นไป และโปรซีเยอร์หมายเลข 0 จะถูกเพิ่มเข้าไปโดยอัตโนมัติในระหว่างการ compile โดย RPCGEN เพื่อให้ใช้ในการทดสอบ (ping) ได้ว่าขณะนั้นโปรเซสผู้ให้บริการยังวิ่งอยู่หรือไม่

การ compile ข้อกำหนดโปรโตคอล จะใช้คำสั่ง rpcgen ดังตัวอย่าง

```
$ rpcgen rdb.x
```

ซึ่งจะได้ไฟล์ stub code ของผู้เรียก ชื่อ rdb_clnt.c ไฟล์ stub code ของผู้ให้บริการ ชื่อ rdb_svc.c ไฟล์ XDR filter code ชื่อ rdb_xdr.c และไฟล์ header ที่ทั้งส่วนผู้ให้บริการ และส่วนผู้ขอบริการจะต้อง #include ชื่อ rdb.h ซึ่งชื่อ procedure ที่กำหนดในไฟล์ rdb.h จะถูกแปลงให้เป็นอักษรตัวเล็ก และต่อท้ายด้วย '_' และ หมายเลข version ซึ่งในที่นี้เท่ากับ 1 ส่วนหมายเลขประจำตัวของแต่ละ procedure ที่จะใช้ในการเรียกได้ถูกกำหนดในรูปแบบ long integer

ส่วน stub ของผู้เรียกที่ได้จาก rpcgen นั้นจะประกอบด้วย procedure ชื่อเดียวกับที่ได้ประกาศไว้ในไฟล์ header นั่นเอง ซึ่งจะเป็นส่วนที่รับการเรียก procedure ปกติจากผู้เรียกแล้วนำ parameter ส่งไปยัง stub ของผู้ให้บริการ โดยใช้ XDR filter ในการแปลงรูปแบบข้อมูลและรอรับ return value จากผู้ให้บริการ โดยใช้ XDR filter ในการแปลงรูปแบบข้อมูล และรอรับ return value จากผู้ให้บริการเพื่อทำการส่งกลับไปให้แก่ผู้เรียกอีกทีหนึ่ง โดยที่ผู้เรียกจะรู้สึกเหมือนว่าเป็นการเรียกใช้ local procedure ธรรมดาเท่านั้น

ส่วน stub ของผู้ให้บริการนั้นจะมีฟังก์ชั่น main() ซึ่งจะทำหน้าที่ register โปรแกรมแล้วรอให้มีผู้เรียก procedure ที่มีอยู่ แล้วจึงแปลง parameter ที่ได้รับในรูป XDR กลับให้อยู่ในรูปเดิม เรียก procedure ที่ต้องการ และส่ง return argument กลับไปให้แก่ผู้เรียกในรูป XDR อีกทีหนึ่ง

2.3.2.2 การเขียน application code ส่วนผู้เรียก และ ผู้ให้บริการ

สำหรับ code ที่ผู้ใช้จะต้องเขียนเองในส่วนผู้เรียกนั้นคือฟังก์ชั่น main() และ local procedure อื่นๆ โดยเมื่อต้องการเรียก remote procedure นั้น ก่อนอื่นจะต้องทำการสร้างการติดต่อกับเครื่องที่มีผู้ให้บริการวิ่งอยู่ และ ขอ register เพื่อตรวจสอบดูว่ามี procedure ที่ต้องการติดต่อหรือไม่ (เปรียบเสมือนการขอเปิดไฟล์ เพื่ออ่าน/เขียน) โดยการเรียกฟังก์ชั่น clnt_create(3N) ซึ่งถ้าไม่มีอะไรผิดพลาดก็จะได้รับ client handler กลับมา (เปรียบเสมือนกับไฟล์ handler) ซึ่งจะต้องใช้ในการเรียก remote procedure ต่อไป

เมื่อได้ client handler มาแล้วก็จะสามารถเรียก remote procedure ได้เหมือนกับการเรียก local procedure ปกติเพียงแต่ต้องใส่ client handle เป็น parameter เพิ่มเติมจาก parameter ที่กำหนดไว้เดิม โดยที่การ

ส่ง parameter ทุกตัวนั้นจะเป็นการส่ง address ของตัวแปรไป และ return value ที่ได้ก็จะเป็น address ของ return value นั้น

ก่อนที่จะจบโปรแกรมก็จะต้องเลิกการติดต่อกับผู้ให้บริการ (เปรียบเหมือนกับการปิดไฟล์) โดยการทำลาย client handler ด้วยฟังก์ชั่น clnt_destroy(3N)

สำหรับ code ในส่วนของผู้ให้บริการที่จะต้องเขียนเองนั้นก็คือส่วนของ procedure ต่างๆที่ได้กำหนดไว้ในโปรโตคอล แล้วนั่นเอง โดยในโปรแกรมตัวอย่างนี้จะให้ชื่อว่า rdb_svc_proc.c

การ compileและ ทดสอบโปรแกรม

2.3.2.3 การ compile และทดสอบโปรแกรม

เมื่อได้ code ครบแล้วก็จะ compile ส่วนของผู้เรียก และผู้ให้บริการ แล้ว link กับ stub และ XDR filter เพื่อให้ได้ executable program สองไฟล์ คือ rdb และ rdb_svc ดังนี้

compile XDR filter : $ cc -c rdb_xdr.c

complile ,link and make client program :

$ cc -o rdb rdb_xdr.o rdb_clnt.c rdb.c

compile ,link and make server program :

$ cc -o rdb_svc rdb_xdr.o rdb_svc.c rdb_svc_proc.c

ทดสอบโดยการ run โปรเซสผู้ให้บริการบนเครื่องใดเครื่องหนึ่งในเครือข่าย โดยทำการ run ในลักษณะที่เป็น back ground ดังนี้

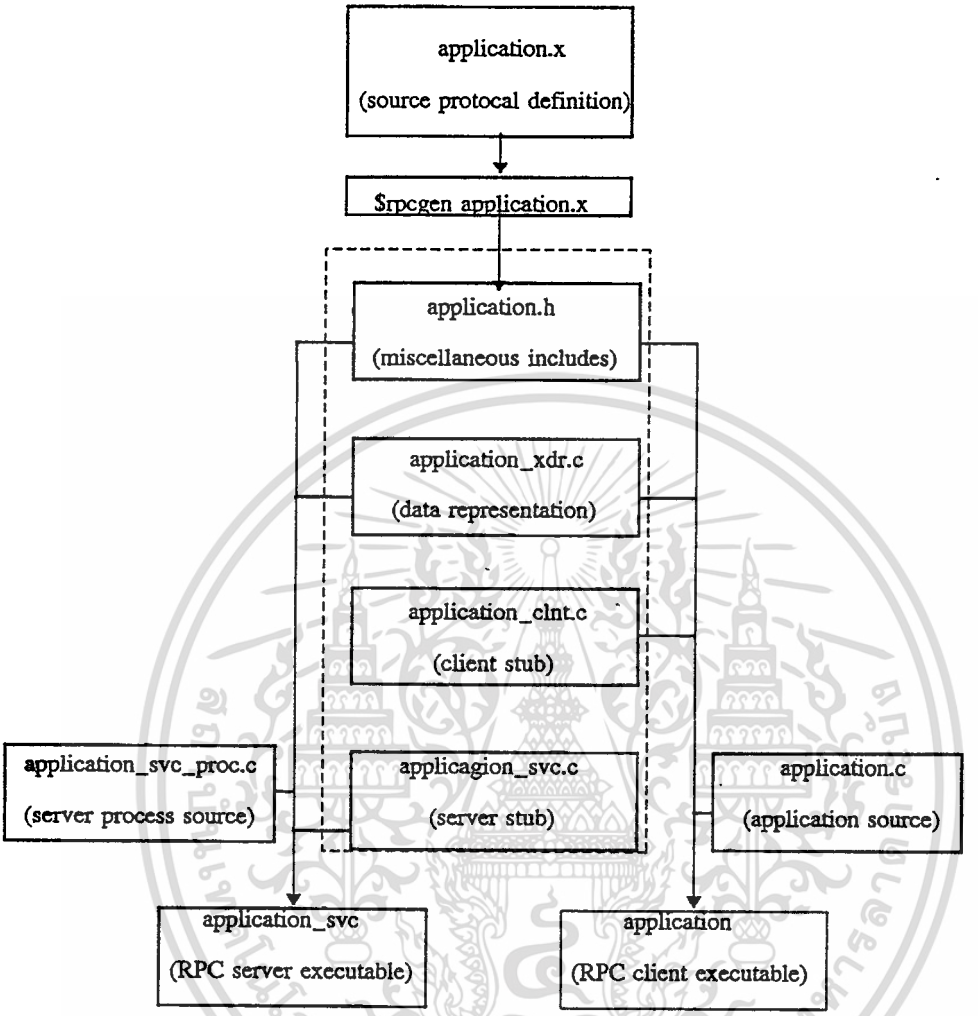
$ rdb_svc &

แล้ว run โปรเซสผู้เรียกที่เครื่องอื่นๆ แล้วตรวจดูความถูกต้องของผลที่ได้

```
$ rdb lcad00 2 SOMCHAI
first    middle    last      phone    location

SOMCHAI  S         KEMKLUD   2734411  LADKRABANG
```

สรุปขั้นตอนในการคอมไพล์โปรแกรมภาษา RPCL ด้วยแผนผังดังนี้



รูปที่ 2.3-4 ขั้นตอนการคอมไพล์โปรแกรมภาษา RPC

2.4 การพัฒนาแอพพลิเคชันบนยูนิกซ์เท็กซ์โหมด

โปรแกรมจัดการจอภาพเป็นส่วนประกอบพื้นฐานของแอพพลิเคชันธุรกิจต่าง ๆ โปรแกรมเหล่านี้จะจัดการอินพุทและเอาท์พุททางจอภาพ โปรแกรมจอภาพอาจเคลื่อนย้ายเคอร์เซอร์, พิมพ์เมนู, แบ่งพื้นที่จอภาพเป็นหน้าต่าง หรือ แสดงจอภาพเพื่อช่วยการป้อนข้อมูลของผู้ใช้ และเรียกข่าวสารจากดาต้าเบส

2.4.1 curses

curses(3X) คือไลบรารีของรูทีนที่สามารถใช้เพื่อเขียนโปรแกรมจัดการจอภาพบนระบบยูนิกซ์ รูทีนเหล่านี้เป็นฟังก์ชันภาษาซีและมาโคร หลายรูทีนก็ทำงานคล้ายคลึงในไลบรารีของภาษาซีมาตรฐาน ตัวอย่างเช่น มีรูทีน printw() ซึ่งทำงานคล้ายกับ printf(3S) มาก และอีกรูทีนคือ getch() ซึ่งมีพฤติกรรมคล้ายกับ getc(3S)

โปรแกรมสำหรับเครื่อง ATM ที่คุณเคยใช้อาจใช้ printw() เพื่อพิมพ์เมนูและใช้ getch() เพื่อรับคำสั่งฝากถอนเงินก็ได้ รวมทั้งโปรแกรมเอดดิเตอร์ของระบบยูนิกซ์ เช่น เอดดิเตอร์ vi(1) อาจใช้รูทีนเหล่านี้และรูทีนอื่นของ curses ก็ได้เช่นกัน

รูทีนของ curses ปกติจะอยู่ใน /usr/lib/libcurses.a เมื่อคุณต้องการใช้รูทีนเหล่านี้คุณต้องคอมไพล์โปรแกรมของคุณด้วยคำสั่ง cc(1) และอินคลูด -lcurses ในบรรทัดคำสั่งเพื่อลิงค์ไปยังเอดดิเตอร์ที่ทำงานและโหลดรูทีนขึ้นมา

```
cc file.c -lcurses -o file
```

ชื่อ curses นี้มาจากคำว่า cursor optimization ซึ่งรูทีนในไลบรารีนี้ทำงานดังชื่อนี้ โดย cursor optimization จะลดจำนวนของเคอร์เซอร์ที่จะต้องวิ่งไปทั่วจอภาพเพื่ออัพเดทจอภาพ ตัวอย่างเช่น ถ้าคุณออกแบบโปรแกรมเอ็ดดิเตอร์ด้วยรูทีน curses และต้องการแก้ไขข้อความ

```
curses/terminfo is a great package for creating screens.
```

ให้เป็น

```
curses/terminfo is the best package for creating screens.
```

โปรแกรมจะส่งข้อความออกมาเพียง the best ในตำแหน่งของ a great ส่วนตัวอักษรอื่น ๆ ก็จะรักษาไว้คงเดิม จึงเห็นได้ว่า cursor optimization จะลดจำนวนข้อมูลที่ต้องส่งได้เป็นอย่างมาก

cursor optimization ช่วยดูแลการอัพเดทจอภาพให้เหมาะสมกับเทอร์มินอล (terminal) ที่ curses รันอยู่ได้ หมายความว่าไลบรารี curses สามารถทำงานได้กับเทอร์มินอลหลายชนิด มันจะค้นหาดาต้าเบส terminfo เพื่อหาลักษณะที่ถูกต้องของเทอร์มินอล

เราจะมาดูกันว่า cursor optimization สามารถช่วยผู้เขียนโปรแกรมและผู้ใช้ได้อย่างไร ประการแรก มันช่วยประหยัดเวลาที่ใช้อธิบายในโปรแกรมว่าคุณต้องการอัพเดทจอภาพอย่างไร ประการที่สอง คือ มันช่วยประหยัดเวลาอัพเดทจอภาพขณะที่ผู้ใช้ใช้งาน ประการที่สาม คือ มันช่วยลดภาระงานบนสายสื่อสารของระบบยูนิกซ์ที่ใช้เพื่ออัพเดท และประการที่สี่ คือ ผู้เขียนโปรแกรมไม่ต้องกังวลเกี่ยวกับเทอร์มินอลหลากหลายชนิดที่โปรแกรมที่จะต้องรัน

รูป 2.4-1 เป็นโปรแกรม curses ง่าย ๆ โปรแกรมนี้ใช้รูทีนพื้นฐานของ curses เพื่อย้ายเคอร์เซอร์ไปยังกลางจอเทอร์มินอล

```
#include <curses.h>
main()
{
        initscr();

        move(LINES/2 - 1, COLS/2 - 4);
        addstr("Bulls");
        refresh();
        addstr("Eye");
        refresh();
        endwin();
}
```

รูป 2.4-1 โปรแกรม curses

2.4.2 การใช้งาน curses

ชื่อ

curses() - แพ็คเกจเพิ่มประสิทธิภาพและจัดการจอภาพ CRT

SYSNOPSIS

```
#include <curses.h>

cc [flags] file ... -lcurses [libraries]
```

คำอธิบาย

รูทีนเหล่านี้เสนอวิธีการอัพเดทจอภาพได้อย่างมีประสิทธิภาพ ก่อนที่จะใช้งานรูทีนใด ๆ ที่เกี่ยวกับหน้าต่างหรือจอภาพ จำเป็นต้องอินนิเชียลรูทีน curses เสียก่อนด้วยคำสั่ง initscr() และก่อนจะออกจากโปรแกรมต้องใช้คำสั่ง endwin() และเมื่อต้องการรับตัวอักษรครั้งละหนึ่งตัวโดยไม่แสดงตัวอักษรให้เห็น (พบมากในโปรแกรมติดต่อกับผู้ใช้ที่เป็น interactive) หลังจากที่เรียก initscr() แล้ว โปรแกรมควรใช้คำสั่ง

```
nonl(); cbreak(); noecho();
```

การใช้อินเทอร์เฟซของ curses อย่างสมบูรณ์ อนุญาตให้จัดการกับโครงสร้างข้อมูลแบบหน้าต่าง (windows) ซึ่งเป็นเหมือนอาร์เรย์สองมิติที่แทนการแสดงผลทั้งจอภาพหรือบางส่วนของจอภาพ ซึ่งมีค่าดีฟอลท์ของหน้าต่างที่เรียกว่า stdscr เตรียมให้อยู่แล้ว และสามารถสร้างหน้าต่างอื่น ๆ อีกโดยใช้คำสั่ง newwin ก่อนที่หน้าต่างจะถูกใช้งานจะต้องประกาศชนิดตัวแปรเป็น "WINDOW * " ชนิดตัวแปร WINDOW นี้ถูกกำหนดไว้

โดยโครงสร้างภาษาซี โครงสร้างของข้อมูลเหล่านี้จะถูกจัดการโดยฟังก์ชันดังที่จะกล่าวต่อไป เช่น ฟังก์ชันที่เป็นพื้นฐานที่สุด เป็นต้นว่า move() และ addch() (เวอร์ชันทั่วไปจะมีฟังก์ชันเหล่านี้ ถ้าชื่อขึ้นต้นด้วย w จะใช้เพื่อจัดการกับหน้าต่าง แต่สำหรับ stdscr ไม่จำเป็นต้องขึ้นต้นด้วย w) หลังจากนั้นต้องเรียก refresh() เพื่อบอกให้รูทีนสร้างจอภาพให้แสดงผลใหม่

Mini-curses เป็นซับเซ็ต (subset) ของ curses ซึ่งไม่อนุญาติให้จัดการกับหน้าต่างมากกว่าหนึ่งหน้าต่าง การเรียกเซ็ตย่อยนี้ ต้องเพิ่มออปชัน -DMINICURSES ในคำสั่งของ cc(1) ซับเซ็ตนี้เล็กกว่าและทำงานได้เร็วกว่า curses แบบเต็ม

ถ้าตัวแปรสภาวะแวดล้อม TERMINFO ถูกกำหนดไว้ โปรแกรมต่าง ๆ ที่ใช้ curses จะต้องตรวจข้อมูลที่กำหนดเป็นโลคอลเทอร์มินอลก่อนตรวจในพื้นที่มาตรฐาน ยกตัวอย่างเช่น ถ้าพื้นที่มาตรฐานคือ /usr/lib/terminfo และ TERM ถูกเซ็ตเป็น vt100 ไฟล์ที่ถูกคอมไพล์จะพบได้ใน /usr/lib/terminfo/v/vt100 (ตัวอักษร v ดึงมาจากตัวอักษรตัวแรกของคำว่า vt100 เพื่อหลีกเลี่ยงการสร้างไดเร็กทอรีขนาดใหญ่) แต่อย่างไรก็ตาม ถ้า TERMINFO ถูกเซ็ทไปยัง /usr/mark/myterms curses จะเริ่มตรวจใน /usr/lib/myterms/v/vt100 ก่อน ถ้าหากไม่พบก็จึงจะไปตรวจใน usr/lib/terminfo/v/vt100 ลักษณะเช่นนี้เป็นประโยชน์แก่การพัฒนาการกำหนดตามต้องการ หรือเมื่อไม่กำหนดเพอร์มิชชั่น (permission) การเขียนให้เขียนใน /usr/lib/terminfo ให้

ฟังก์ชัน

ฟังก์ชันที่แสดงในตาราง 2.4-1 นี้เป็นเพียงส่วนหนึ่งของฟังก์ชันที่มีทั้งหมดของ curses ฟังก์ชันเหล่านี้เป็นฟังก์ชันบางฟังก์ชันที่ใช้ในการพัฒนาโครงงานระบบจำลองการซื้อขายหลักทรัพย์ ฟังก์ชันอื่น ๆ สามารถหาเพิ่มเติมได้ในคู่มืออ้างอิงการเขียนโปรแกรมบนยูนิกซ์ และในไฟล์ help ของระบบยูนิกซ์ รายชื่อฟังก์ชันที่แสดงนี้เรียงลำดับตามตัวอักษร ฟังก์ชันที่มีเครื่องหมายดอกจันเป็นฟังก์ชันที่สามารถเรียกได้โดย Mini-curses

ชื่อฟังก์ชัน	ความหมาย
box(win, vert, hor)	วาด box รอบของของ win โดย vert และ hor เป็นตัวอักษรที่ใช้เป็นขอบตามแนวตั้งและแนวนอนของ box
clear()	ลบข้อความทั้งหมดบน stdscr
delwin(win)	ทำลาย win
echo()*	เซ็ทโหมด echo
endwin()*	จบโหมด window
initscr()*	อินนิเชียลจอภาพ
keypad(win, bf)	ทำให้ keypad ของอินพุททำงาน
newwin(lines, cols, begin y, begin x)	สร้าง window ใหม่
nl()*	เซ็ทการขึ้นบรรทัดใหม่

noecho()*	เซ็ทให้โหมด echo ไม่ทำงาน
nonl()*	เซ็ทให้การขึ้นบรรทัดใหม่ไม่ทำงาน
waddstr(win, str)	เพิ่มประโยคให้ win
wattroff(win, attrs)	ปิดการแสดงตามแบบตัวอักษร attrs ใน win
wattron(win, attrs)	เปิดการแสดงตามแบบตัวอักษร attrs ใน win
wclear(win)	ลบข้อความทั้งหมดใน win
werase(win)	ทำลาย win
wgetch(win)	รับตัวอักษรผ่าน win
winsertln(win)	แทรกบรรทัดใน win
wmove(win, y, x)	เซ็ทโคออร์ดิเนทปัจจุบัน (y, x) บน win
wprintw(win, fmt, arg1, arg2, ...)	printf() บน win
wrefresh(win)	ทำจอภาพให้เป็นเหมือน win
wscanw(win, fmt, arg1, arg2, ...)	scanf() ผ่าน win

ตาราง 2.4-1 บางฟังก์ชันของ curses

แอททริบิวท

ค่าวีดีโอแอททริบิวท์ในตาราง 2.4-2 ต่อไปนี้สามารถผ่านไปยังฟังก์ชัน attron(), attroff() และ attrset()

ชื่อแอททริบิวท์	ความหมาย
A_STAANDOUT	โหมดที่เห็นได้ชัดเจนที่สุดของเทอร์มินอล
A_UNDERLINE	ขีดเส้นใต้
A_REVERSE	แสดงกลับสีของวีดีโอ
A_BLINK	กระพริบ
A_DIM	สว่างปานกลาง
A_BOLD	สว่างมาก หรือ ตัวหนา
A_BLANK	ว่าง (ไม่สามารถมองเห็น)
A_PROTECT	ป้องกัน
A_ALTCHARSET	ชุดตัวอักษรที่เลือก

ตาราง 2.4-2 ตารางแสดงแอททริบิวท์ใน curses

ฟังก์ชันคีย์

ฟังก์ชันคีย์เหล่านี้ใช้เพื่อการติดต่อกับผู้ใช้ เช่น ในเมนู ฟังก์ชันคีย์สามารถถูกรีเทิร์นจาก getch เมื่อให้ keypad ทำงาน ซึ่งค่าที่แสดงเหล่านี้อาจไม่มีเมื่อกำหนด terminfo บางค่า หรือ เทอร์มินอลอาจส่งค่าของโค๊ดที่ไม่ตรงกันเมื่อคีย์ถูกกด

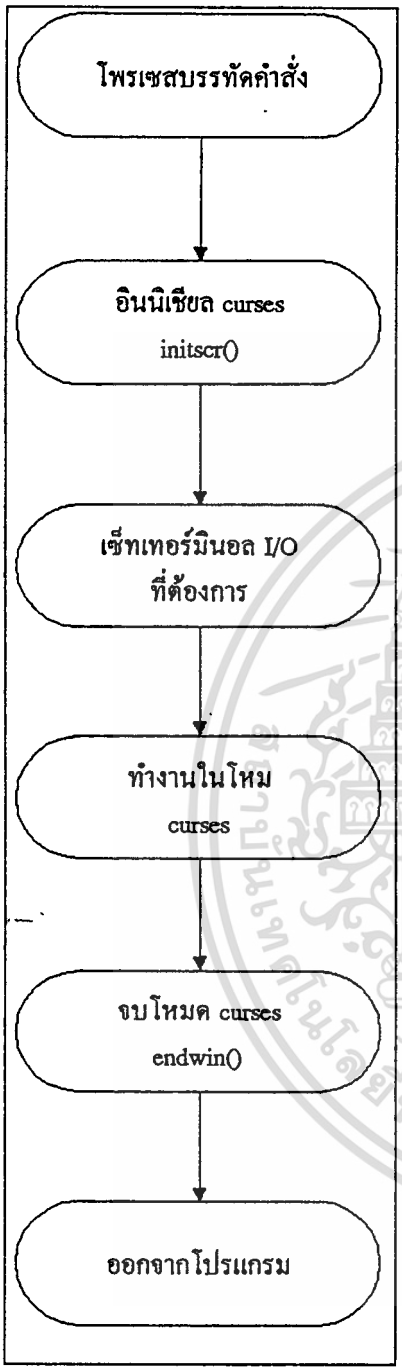
ฟังก์ชันคีย์ที่แสดงนี้ เป็นฟังก์ชันคีย์ที่มีใช้ในโครงงาน ฯ เท่านั้น รายการของฟังก์ชันคีย์อื่น เช่น KEY_BREAK และ KEY_BACKSPACE มีปรากฏในคู่มืออ้างอิงสำหรับการเขียนบนยูนิกซ์ และในไฟล์ help ของยูนิกซ์

ชื่อ	ค่า	ชื่อคีย์
KEY_DOWN	0402	คีย์ลูกศรลง
KEY_UP	0403	คีย์ลูกศรขึ้น
KEY_LEFT	0404	คีย์ลูกศรไปซ้าย
KEY_RIGHT	0405	คีย์ลูกศรไปขวา

ตาราง 2.4-3 ฟังก์ชันคีย์บางส่วนที่ใช้ในโครงงาน

2.4.3 โครงสร้างพื้นฐานของโปรแกรม curses

โครงสร้างพื้นฐานของโปรแกรม curses มีดังนี้



ก่อนที่จะเริ่มใช้ curses เราควรทำการโพรเซสบรรทัดคำสั่งเสียก่อน ยกตัวอย่างเช่น ถ้าผู้ใช้ป้อนอาร์กิวเมนต์ในบรรทัดคำสั่งไม่ถูกต้อง โปรแกรมควรจะพิมพ์ข้อความไปยัง stderr โดยใช้ fprintf(3) และออกจากโปรแกรมไม่ต้องเข้าสู่โหมด curses จนกว่าโปรแกรมจะพร้อม

โครงสร้างภายในและตัวแปรหลาย ๆ ตัว จำเป็นต้องถูกตั้งค่าก่อนที่ curses จะถูกใช้ ดังนั้น จะต้องอินนิเชียล curses โดยปกติจะใช้คำสั่ง initscr() เพื่อเตรียมหน้าต่าง curscr และ stdscr , อินนิเชียลเทอร์มินอล, และโปรแกรมจึงเข้าสู่โหมด in-curses

ถ้าคุณต้องการตั้งค่าโหมด I/O พิเศษในเทอร์มินอลไดร์เวอร์ คุณต้องทำหลังจากอินนิเชียลแล้ว โดยปกติแล้วโหมดเหล่านี้จะถูกเซ็ทเพียงครั้งเดียว และจำยังคงไม่แก้ไขอีกตลอดเวลาที่โปรแกรมทำงาน ยกตัวอย่างเช่น การเซ็ทเทอร์มินอลให้เข้าสู่โหมด raw, ปิดการ echoing และอื่น ๆ

ขณะนี้โปรแกรมทำงานในโหมด curses และทำงานต่อไปได้โดยใช้เครื่องมือของ curses ยกตัวอย่างเช่น สร้าง/ทำลาย หน้าต่าง, เพิ่มตัวอักษรในหน้าต่าง, รับอินพุทจากคีย์บอร์ด และอื่น ๆ

เมื่อโพรเซสสมบูรณ์ และโปรแกรมพร้อมที่จะรีเทิร์นกลับไปยังเชลล์ (หรือโพรเซสที่เรียก) เทอร์มินอลจะต้องกลับไปยังโหมดการทำงานดั้งเดิม คือ โหมดปัจจุบันก่อนที่จะเข้ามาในโปรแกรม ขั้นตอนนี้ทำโดย endwin()

และในที่สุด ก็ออกจากโปรแกรม

รูปที่ 2.4-2 โครงสร้างพื้นฐานของโปรแกรม curses

2.5 ความรู้เรื่องตลาดหลักทรัพย์

2.5.1 องค์ประกอบของตลาดหลักทรัพย์

ตลาดหลักทรัพย์ประกอบด้วย

ผู้ลงทุนอาจแบ่งเป็น นักเก็งกำไร ผู้ลงทุนระยะสั้นและผู้ลงทุนระยะยาว

นักเก็งกำไร เป็นบุคคลที่เสี่ยงหวังผลกำไรจากหลักทรัพย์ในช่วงสั้น มักไม่คำนึงถึงเงินปันผลที่จะได้รับ อาจมีการซื้อขายในวันเดียวกันก็ได้ถ้ามีกำไร

ผู้ลงทุนระยะสั้น สนใจผลตอบแทนในรูปของกำไรจากการขายเป็นสำคัญและเงินปันผลบ้าง แต่มักระวังเกี่ยวกับราคาที่อาจลดลงอันเนื่องมาจากภาวะตลาด

ผู้ลงทุนระยะยาว หวังผลตอบแทนในระยะยาว ได้แก่ ดอกเบี้ย เงินปันผล รวมทั้งสิทธิต่าง ๆ ของหุ้น ตลอดจนกำไรจากการขายหลักทรัพย์ มักมีความตั้งใจจะถือหุ้นเกินกว่า 3 ปี

2.5.2 ขั้นตอนการซื้อขายหลักทรัพย์

การซื้อขายหลักทรัพย์ในตลาดหลักทรัพย์แห่งประเทศไทย แบ่งขั้นตอนออกได้ดังนี้

1. การตัดสินใจ กรณีจะซื้อหุ้น ผู้ซื้อจะต้องพิจารณาวิเคราะห์การดำเนินงานของบริษัทจดทะเบียนหรือบริษัทรับอนุญาตที่ตนมีความประสงค์จะซื้อในแง่การให้ผลตอบแทนการลงทุนว่าคุ้มไหม ส่วนกรณีการขายหุ้น ผู้ถือหุ้นอาจขายเพราะมีความจำเป็นต้องใช้เงิน หรือขายเพราะตนได้ซื้อมาในราคาถูกกว่าเพื่อเอากำไรก็ได้ ขั้นตอนแรกนี้เป็นขั้นตอนที่เรียกว่า "การตัดสินใจ"

2. การสั่งซื้อขาย เมื่อตัดสินใจว่าจะซื้อหรือขายแล้ว ก็จะติดต่อกับบริษัทสมาชิก (นายหน้า) เพื่อสั่งให้ดำเนินการซื้อขายหุ้นให้ตามจำนวนและราคาที่ต้องการ

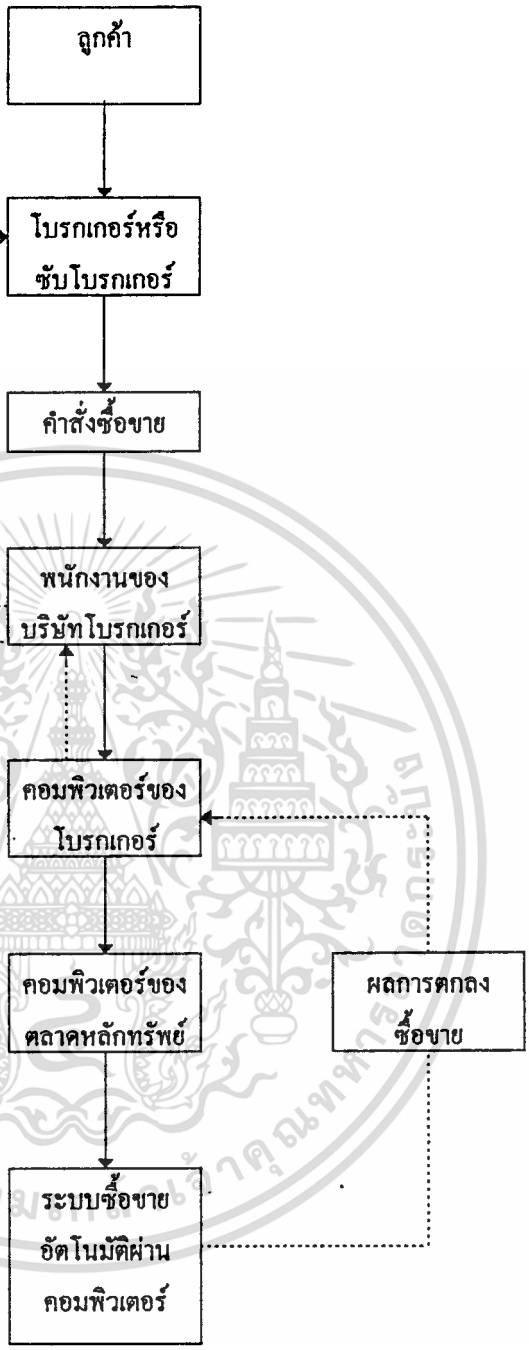
3. การส่งคำสั่ง คำสั่งที่ลูกค้าสั่งไว้ที่บริษัทสมาชิกตามข้อ 2 จะถูกส่งเข้าไปยังเจ้าหน้าที่ของบริษัทสมาชิกนั้นที่ปฏิบัติงานอยู่ในห้องค้าหลักทรัพย์ เพื่อให้เจ้าหน้าที่ทำการตกลงซื้อขายตามคำสั่งนั้น ๆ ต่อไป

4. การตกลงซื้อขาย กรณีที่คำสั่งนั้นสามารถตกลงซื้อขายได้ทันที เจ้าหน้าที่ของบริษัทสมาชิกก็จะตกลงซื้อขายได้ แต่ถ้ายังตกลงซื้อขายไม่ได้ เจ้าหน้าที่ของบริษัทสมาชิกก็จะจดแจ้งคำเสนอไว้ตามที่ลูกค้าสั่งเพื่อรอการตกลงการซื้อขาย

5. การทำหลักฐานการตกลง เมื่อเจ้าหน้าที่ของบริษัทสมาชิกตกลงซื้อขายได้ตามคำสั่งนั้นแล้ว เจ้าหน้าที่บริษัทสมาชิกผู้ขายจะบันทึกรายการซื้อขายนั้นไว้ในแบบบันทึกรายการซื้อขายใต้กระดานของหลักทรัพย์นั้น โดยเจ้าหน้าที่บริษัทสมาชิกผู้ซื้อและเจ้าหน้าที่สมาชิกผู้ขายก็จะจัดทำสัญญาซื้อขายนั้น ซึ่งมี 3 คู่ฉบับ เจ้าหน้าที่บริษัทสมาชิกผู้ซื้อและเจ้าหน้าที่บริษัทสมาชิกผู้ขายลงลายมือชื่อไว้แล้ว เจ้าหน้าที่บริษัทสมาชิกผู้ขายจะส่งคู่ฉบับหนึ่งให้เจ้าหน้าที่บริษัทสมาชิกผู้ซื้อเก็บไว้ ส่งอีกคู่ฉบับหนึ่งให้เจ้าหน้าที่ตลาดหลักทรัพย์ ส่วนคู่ฉบับที่ 3 เจ้าหน้าที่บริษัทสมาชิกผู้ขายเก็บไว้เอง

6. ความสมบูรณ์ของการตกลง เมื่อได้ปฏิบัติตามขั้นตอนที่ 1 ถึง 5 แล้ว ถือว่าการซื้อขายรายการนั้นเสร็จสิ้นสมบูรณ์ในขั้นตอนการซื้อขาย และจะยกเลิกการซื้อขายรายการที่เกิดขึ้นแล้วนี้ไม่ได้ไม่ว่ากรณีใด ๆ

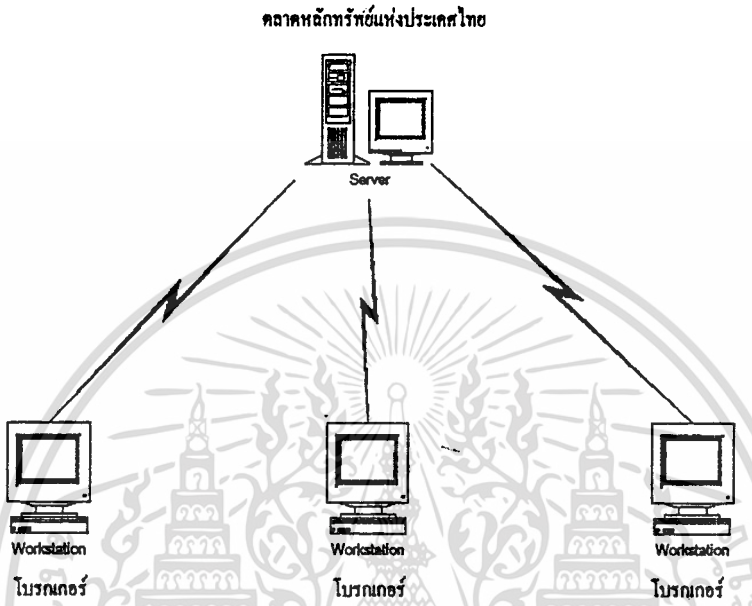
การซื้อขายหลักทรัพย์ในปัจจุบันผ่านระบบอัตโนมัติในคอมพิวเตอร์ของตลาดหลักทรัพย์แห่งประเทศไทย ซึ่งเราสามารถแสดงเป็นแผนภาพขั้นตอนการซื้อขายได้ดังนี้



รูปที่ 2.5-1 ขั้นตอนการซื้อขายหลักทรัพย์

เมื่อลูกค้าตัดสินใจซื้อหรือขายหลักทรัพย์ของตน ลูกค้าจะต้องแจ้งความประสงค์แก่โบรกเกอร์หรือซับโบรกเกอร์ของตน จากนั้นโบรกเกอร์หรือซับโบรกเกอร์จะส่งคำสั่งซื้อขายที่จัดทำขึ้นตามความต้องการของลูกค้าไปให้แก่พนักงานของบริษัทโบรกเกอร์ เพื่อส่งข้อมูลเข้าไปยังคอมพิวเตอร์ของตลาดหลักทรัพย์ด้วยคอมพิวเตอร์ในบริษัทตน คำสั่งซื้อขายนี้ก็จะเข้าสู่ระบบซื้อขายอัตโนมัติที่ทำงานโดยคอมพิวเตอร์ทั้งหมด เมื่อการซื้อขายสำเร็จ ระบบคอมพิวเตอร์ควบคุมการซื้อขายของตลาดหลักทรัพย์จะส่งผลการซื้อขายหลักทรัพย์

กลับไปยังคอมพิวเตอร์ของบริษัทโบรกเกอร์ เพื่อให้พนักงานของบริษัทแจ้งข่าวการซื้อขายกลับไปยังลูกค้าของตนอีกครั้งหนึ่ง



รูปที่ 2.5-2 ระบบจำลองตลาดหลักทรัพย์แห่งประเทศไทย

โครงงานนี้ เป็นการพัฒนาแอพพลิเคชัน เพื่อจำลองระบบการทำงานของการซื้อขายหลักทรัพย์ในตลาดหลักทรัพย์ ครอบคลุมทั้งในส่วนของบริษัทโบรกเกอร์ต่าง ๆ และตลาดหลักทรัพย์แห่งประเทศไทย กล่าวคือ การรันแอพพลิเคชันจำเป็นต้องมีโฮสต์อย่างน้อยสองตัวขึ้นไป เพื่อให้ตัวหนึ่งทำหน้าที่เปรียบเสมือนตลาดหลักทรัพย์ และโฮสต์ที่เหลือทำหน้าที่เป็นบริษัทโบรกเกอร์ต่าง ๆ ที่ต้องการขอใช้บริการระบบซื้อขายอัตโนมัติของตลาดหลักทรัพย์

บทที่ 3
การออกแบบและโครงสร้างของโปรแกรม

3.1 โครงสร้างการทำงานโดยรวม

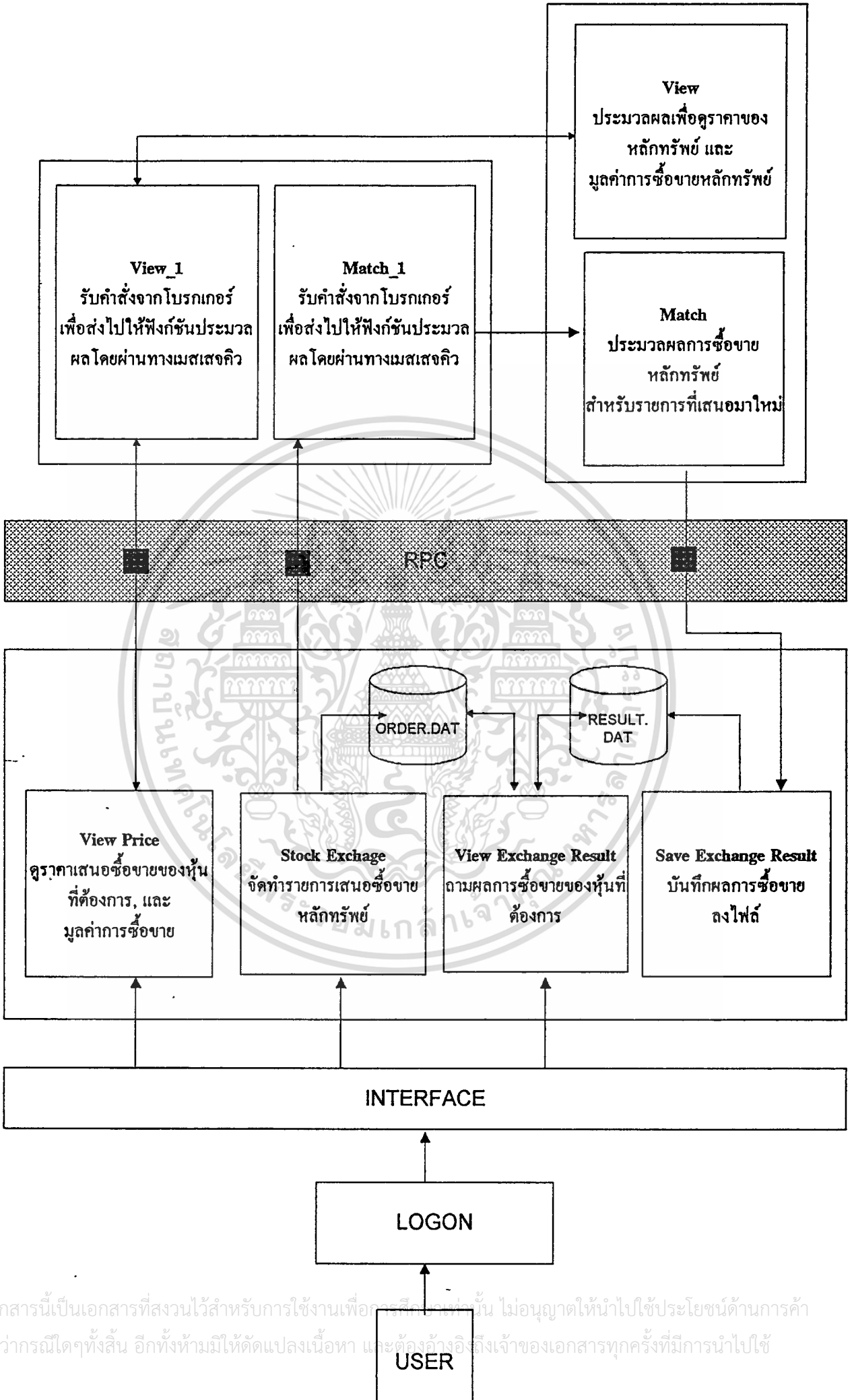
โครงสร้างของระบบจำลองการซื้อขายหลักทรัพย์ สามารถแสดงได้ในรูปของฟังก์ชั่นที่สามารถเรียกใช้งานได้ และ ระบบการติดต่อของโปรเซสจะแบ่งออกเป็นสองส่วนคือ การติดต่อระหว่างบริษัทนายหน้าค้าหลักทรัพย์ หรือที่เรียกกันว่า บริษัท โบรกเกอร์ และ ตลาดหลักทรัพย์ ในโครงงานนี้จะกำหนดให้โบรกเกอร์ และตลาดหลักทรัพย์ อยู่คนละ โฮสต์ (host) กัน ตลาดหลักทรัพย์จะเป็นศูนย์กลางรับคำสั่งซื้อขาย และ คำสั่งการประมวลผลจาก หลาย ๆ โบรกเกอร์ ที่อยู่ต่างโฮสต์ ในช่วงเวลาเดียวกัน โดยใช้วิธีการติดต่อแบบ RPC ตลาดหลักทรัพย์จะทำการตัดราคาซื้อขายหลักทรัพย์ให้กับโบรกเกอร์ โดยจะรับข้อมูลคำสั่งซื้อขาย มาร่วมประมวลผลในที่เดียวกัน

ลักษณะการทำงานเช่นนี้จัดว่าเป็นการทำงานแบบ client server แบบ การกระจายการทำงานของแอพพลิเคชัน (Distributed Logic) ระบบนี้มีการทำงานการกระจายการทำงานของตัวแอพพลิเคชัน จะมีการประมวลผลทั้งทางด้าน ไคลเอนต์ และ เซอร์ฟเวอร์ โดยทางด้านบริษัทโบรกเกอร์ จะเป็นเสมือนตัว ไคลเอนต์ ทำหน้าที่ในการ แสดงการอินเทอร์เฟส สำหรับติดต่อกับผู้ใช้จะมีการรับคำสั่ง และแสดงผลที่โบรกเกอร์แต่ละตัวเอง โบรกเกอร์จะมีการตรวจสอบความถูกต้องของข้อมูลที่จะผู้ใช้ป้อนด้วย ก่อนที่จะส่งคำสั่งไปทำการประมวลผลที่ตลาดหลักทรัพย์ ตลาดหลักทรัพย์จะทำหน้าที่เป็น เซอร์ฟเวอร์ รับคำสั่งซื้อ-ขายหลักทรัพย์ จะการบริษัทโบรกเกอร์ และนำมาประมวลผลกับข้อมูลส่วนกลางที่มีอยู่ และแจ้งผลกลับไปให้โบรกเกอร์ที่เสนอ การซื้อ-ขาย หลักทรัพย์ เมื่อทำการซื้อขายได้สำเร็จ จะแสดงของเขตของระบบจำลองการซื้อหลักทรัพย์ ได้ดังรูป 3.1-1

3.2 โครงสร้างโปรเซส

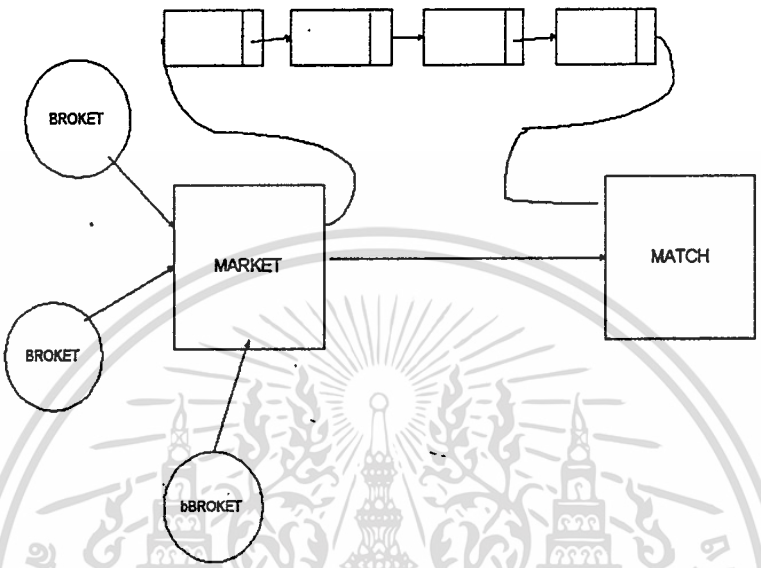
3.2.1 การออกแบบการติดต่อระหว่างโปรเซส

โปรเซสในการทำคำสั่งซื้อขายและขอดูข้อมูลหุ้น สาเหตุที่เลือกการออกแบบ การติดต่อระหว่างโปรเซสในรูปแบบของ RPC และ message queue มีดังนี้



เนื่องจากระบบการซื้อขายหลักทรัพย์ จะมีคำสั่งที่ส่งมาจากโบรกเกอร์ จากหลายๆ host เราอาจมองว่า host ต่างเหล่านี้ประพฤติตัวเป็น client ข้อมูลทั้งหมดจะเข้ามาประมวลผลรวมกันที่เดียวคือ ที่ตลาดหลักทรัพย์ หรืออีกนัยหนึ่งเป็น server ที่ทำหน้าที่ประมวลผล โดยการเรียกฟังก์ชั่นแบบ RPC ในการเรียกฟังก์ชั่นแต่ละครั้ง ระบบทางตลาดหลักทรัพย์จะทำการ fork โปรเซสขึ้นมาใหม่ เพื่อจัดการทำงานคำสั่งนั้น

พิจารณารูปที่ 3.2.1 -1 ประกอบ



รูปที่ 3.2.1 -1 แสดงการติดต่อระหว่างโปรเซส

วิธีในการทำงานร่วมกันของโปรเซสเหล่านี้เป็นไปได้ 2 วิธีคือ

1. การทำ Share memory และ semaphore

เป็นวิธีการเข้าใช้พื้นที่ข้อมูลส่วนรวม และในขณะใดขณะหนึ่งจะมีโปรเซสที่ทำการเปลี่ยนแปลง แก้ไขข้อมูลส่วนรวมได้ 1 โปรเซสเท่านั้น

วิธีการใช้ share memory นี้ แต่ละโปรเซสใช้พื้นที่ใน data regian ร่วมกัน ดังนั้นไม่สามารถใช้คำสั่ง malloc เพื่อจองพื้นที่เพิ่มได้ เนื่องจากว่าพื้นที่ใหม่นั้นอาจจะไปทับกับพื้นที่ที่มีโปรเซสอื่นใช้งานอยู่ ซึ่งการเขียนโปรแกรมในงานระบบการซื้อขายหลักทรัพย์ จำเป็นต้องสร้างพื้นที่เพิ่ม เพื่อเก็บคำสั่งที่ไม่สามารถซื้อขายได้หมดในขณะที่เพิ่งเข้ามาทำการตัดราคา ดังนั้นจึงไม่สามารถใช้วิธี share memory และsemaphore ได้

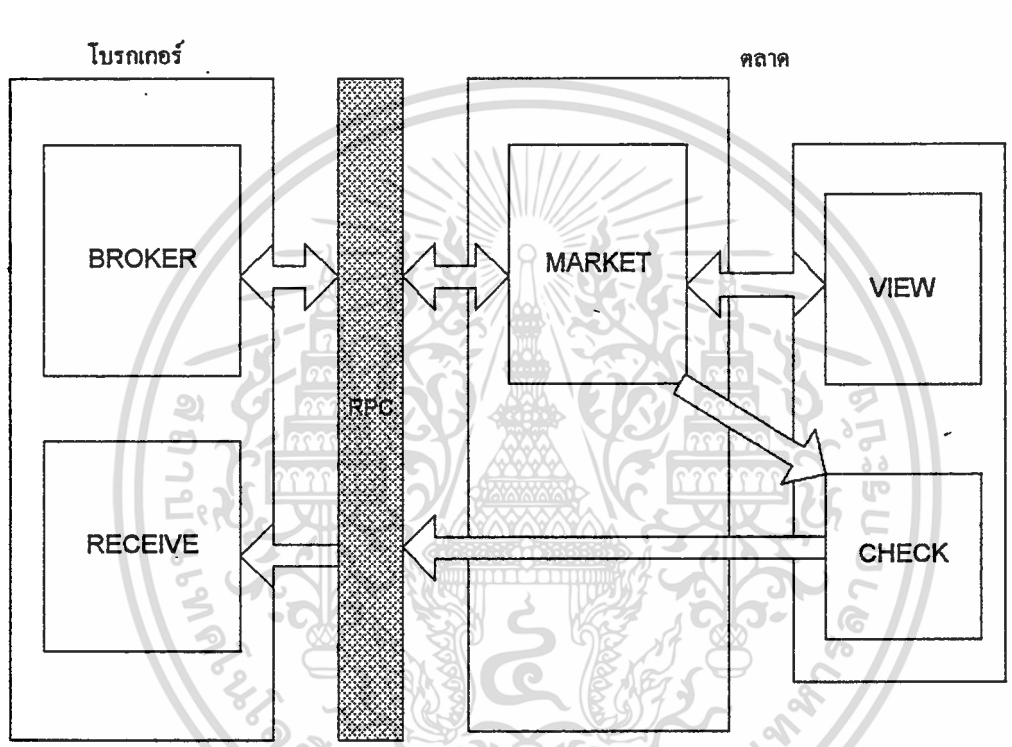
2. การใช้ message queue

ในสภาพแวดล้อมที่มีโปรเซสหลาย ๆ โปรเซสต้องใช้ข้อมูลส่วนกลางร่วมกัน ในโครงงานนี้โปรเซส มีการติดต่อมายังฝั่ง server แบบ RPC นั้นทุกโปรเซสที่ต้องการทำงานอย่างเดียวกัน จะเรียกฟังก์ชั่นเดียวกัน ดังนั้นจะต้องออกแบบวิธีที่สามารถนำคำสั่งเหล่านั้นไปทำงานในพื้นที่ร่วมกัน โดยเลือกวิธีรับคำสั่งจากการเรียกฟังก์ชั่นของโปรเซสต่าง ๆ นำมาสร้างเป็นเมสเซส แล้วส่งไปทำการจัดลำดับการทำงานโดยคิว จะเป็นวิธีในการจัดการการเข้าทำงานฟังก์ชั่นและ ให้พื้นที่ ทีละเมสเซส หรืออีกนัยหนึ่งคือการจัดลำดับการเข้าใช้พื้นที่ทีละโปรเซสนั่นเอง

การทำงานของโปรเซสที่มีฟังก์ชั่นที่ทำการตัดราคาจริง ก็จะรับข้อมูลออกจาก queue ทีละเมสเซส และประมวลผล ภายในพื้นที่ข้อมูลเดียวกัน

3.2.2 โครงสร้างโปรเซสรวมของทั้งระบบ

สำหรับโครงสร้างของโปรเซสในฝั่งโบรกเกอร์ และฝั่งตลาดหลักทรัพย์นั่น สามารถแสดงได้ดังรูปที่ 3.2.2 -1 ซึ่งทางฝั่งโบรกเกอร์นั้นจะมีสองโปรเซสคือ 'broker' และ'receive' ส่วนฝั่งผู้เล่นนั่นจะมีการทำงานเป็นสองส่วนคือ โปรเซส 'market' ทำหน้าที่ในการรับการเรียกฟังก์ชั่นจาก RPC และโปรเซส 'match' ทำหน้าที่ในการทำการจับคู่ราคาคำสั่งเสนอซื้อ เสนอขาย โดยคำสั่งต่างๆ จะถูกนำมาเข้าคิว เพื่อรอการประมวลผลทีละคำสั่ง



รูปที่ 3.2.2 -1 โครงสร้างโปรเซสระบบการซื้อขาย

โปรเซส BROKER และโปรเซส MARKET ติดต่อกันโดยวิธี RPC ทั้งสองโปรเซสจะมองเห็นข้อมูลในการติดต่อร่วมกัน โดย RPC จะสร้างโปรแกรมที่คอยจัดการทางด้านข้อมูลให้ ทั้งทางด้าน โปรเซส BROKER (อยู่ในลักษณะเป็น client) และ โปรเซส MATCH (อยู่ในสภาพ server)

- โปรเซส BROKER เป็นโปรเซสสำหรับรับคำสั่งในรายการต่าง ๆ จากผู้ใช้ และเรียกฟังก์ชั่นของ โปรเซส MARKET ผ่านการติดต่อแบบ RPC

- โปรเซส MARKET จะทำการรันโปรเซส นี้เป็นโปรเซสเบื้องหลัง (back ground) รอการเรียกฟังก์ชั่นของโปรเซสนี้ ผ่านการติดต่อแบบ RPC ฟังก์ชั่นที่สามารถเรียกผ่าน RPC ได้แก่

1. int *req_logon_1() เป็นฟังก์ชั่นสำหรับขอเข้าระบบการซื้อขายหุ้น จะมีการตรวจเช็ค ชื่อโบรกเกอร์ และ รหัสผ่าน และส่งผลลัพธ์กลับไปยังโปรเซสที่เรียกมาทันที

2. void *match_1() เป็นฟังก์ชั่นสำหรับทำคำสั่งซื้อขายหุ้น เมื่อรับพารามิเตอร์แล้ว จะส่งสร้างเป็นเมสเซส และส่งผ่านคิวไปยังโปรเซส match

3. view_price_1() รับคำสั่งขอดูข้อมูลคำสั่งซื้อขายหุ้น และสถิติการซื้อขายหุ้นแต่ละตัว และจะส่งเป็นเมสเซสเข้าไปในคิวเดียวกับ ฟังก์ชัน match_1

4. clear_user_logout_1 ทำการเปลี่ยนแปลงข้อมูล สำหรับตรวจการเข้าและออกจากระบบของโบรกเกอร์ โดยมีข้อกำหนด ให้โบรกเกอร์แต่ละราย ไม่สามารถ เข้าขอใช้ระบบได้ในขณะเดียวกันหลาย ๆ ครั้ง

- โปรเซส MATCH รอรับเมสเซส ที่ผ่านเข้ามาทางคิว และจะแยกชนิดของเมสเซสให้ทำงานตามฟังก์ชั่นที่ต้องการ โดยอาศัยชนิดของโปรเซส mtype เป็นสัญลักษณ์บอกให้ทำงานฟังก์ชั่นใด โปรเซส นี้ประกอบด้วย 2 ฟังก์ชั่นดังนี้

1.ฟังก์ชั่น match ทำการประมวลผลจับคู่ราคาคำสั่งซื้อ ขายที่ส่งมา หลังจากที่มีการซื้อขายสำเร็จจะแจ้งผลลัพธ์กลับไปที่ host ของผู้ที่ส่งคำสั่งทั้งซื้อและขาย การส่งผลลัพธ์กลับโดยการเรียกฟังก์ชั่นของโปรเซส receive ในการติดต่อแบบ RPC

2.ฟังก์ชั่น view จะอ่านและรายงานผลราคาคำสั่ง ซื้อ ขาย ของข้อมูลหุ้น ตามหมายเลขหุ้นที่ผู้ใช้ต้องการ และส่งค่าสถิติของหุ้นตัวนั้น หลังจากที่ได้ผลลัพธ์แล้ว ก็จะสร้างเมสเซสคิวใหม่ และส่งเมสเซสผ่านคิว กลับมายังโปรเซส market จากนั้นจะทำการส่งค่ากลับโดยการรีเทริน กลับไปทาง RPC

3.3 โครงสร้างข้อมูล(Data Struxture)

3.3.1 ข้อมูลในการขออนุญาตเข้าระบบการซื้อขายของตลาดหลักทรัพย์

ก่อนที่โบรกเกอร์แต่ละรายจะเข้าสู่ระบบการซื้อขายหลักทรัพย์ได้นั้น โบรกเกอร์จะต้องทำการขออนุญาตเข้าระบบ โดยการป้อนชื่อของโบรกเกอร์ และ รหัสผ่าน (password) ข้อมูลจะถูกส่งผ่านการติดต่อแบบ rpc ข้อมูลจะถูกเก็บลงโครงสร้างข้อมูลดังนี้

```
struct   log_type
      { char        host <HOSTNAME_LEN>;
        char        name<MAX_STR>;
        char        password<MAX_STR>;
      };
```

- host <HOSTNAME_LEN> : เป็นชื่อของ host ของตลาดหลักทรัพย์ ที่จะขอเข้าระบบการซื้อขาย กำหนดให้เป็นข้อมูลแบบสตริง ขนาด 8 ไบต์

- name<MAX_STR> : เก็บชื่อของโบร่กเกอร์ เป็นข้อมูลแบบสตริง จะมีการตรวจสอบความถูกต้องโดยเทียบค่ากับข้อมูลที่อยในไฟล์ stockname.dat ก่อนที่จะนำตัวแปรนี้เก็บเข้า ตัวแปรแบบโครงสร้างที่จะส่งไป ตรวจสอบทางด้านตลาดหลักทรัพย์
- password<MAX_STR> : เป็นข้อมูลแบบ สตริงขนาด 30 ไบต์ เพื่อเก็บรหัสผ่าน ของโบรกเกอร์แต่ละราย

การตรวจเช็คความถูกต้องของข้อมูล เช่น ชื่อ host และชื่อ โบรกเกอร์ จะตรวจเช็คที่ฝั่ง โบรกเกอร์เอง เพื่อประหยัดเวลา ลดการใช้ทรัพยากรในการส่ง

ทางด้าน server จะทำการตรวจสอบ รหัสผ่านของโบรกเกอร์ จาก ไฟล์ user.dat หลังจากตรวจสอบแล้วจะส่ง ผลกลับดังนี้

-1 ตรวจแล้วไม่พบ ว่ามีรายชื่อของ โบรกเกอร์นั้นอยู่ หรือใส่ข้อมูลผิดพลาด

-2 ข้อมูลถูกต้อง แต่ไม่มีการขอใช้ ชื่อการเข้าระบบ ซ้ำซ้อนกัน

ค่าที่ส่งคืนมานอกเหนือจากนี้ แสดงว่าข้อมูลถูกต้อง และไม่มีการเข้าระบบซ้ำซ้อนกัน

- already_login[count] : เป็นอาเรย์ขนาดหนึ่งมิติ เป็นที่เก็บข้อมูลการเข้าระบบของโบรกเกอร์
  ถ้ายังไม่เคยเข้าระบบ จะเก็บตัวเลข 0
  ขณะนี้มีการขอใช้ระบบอยู่แล้ว 1
  ออกจากระบบ ค่าจะถูกกำหนดกลับเป็น 0
- ไฟล์ user.dat สำหรับเก็บ ข้อมูล ชื่อ และ รหัสผ่าน สำหรับการตรวจสอบโดย ตลาดหลักทรัพย์

3.3.2 ข้อมูลสำหรับโปรเซสการทำการซื้อ-ขายหุ้น

- stock_number เก็บหมายเลขหุ้น ข้อมูลนี้มาจากการรับข้อมูลจาก ผู้ใช้เป็นชนิด สตริง จากนั้นจะทำการอ่านไฟล์ stockname.dat ซึ่งเก็บ รายชื่อหุ้นเรียงตามหมายเลขของหุ้นไว้ จะทำการหาชื่อหุ้นและ เปลี่ยนแปลงให้อยู่ในค่าจำนวนเต็ม ชื่อ stock_number จะใช้หมายเลขหุ้นในการส่งไปประมวลผล
- bid_offer : เป็นสถานะในการซื้อขายว่าจะทำการซื้อ (bid) หรือทำคำสั่งขาย(offer) ผู้ใช้จะใส่หมายเลขแทนคำสั่งที่หน้าจอดังนี้ ต้องการเสนอซื้อ จะใส่หมายเลข 0 เสนอขายใส่หมายเลข 1
- price : เป็นราคาที่เสนอของคำสั่งซื้อ-ขาย หน่วยเป็นจำนวน (บาท)
- value : เป็นจำนวนหุ้นที่ต้องการเสนอซื้อ หรือ ขาย ในแต่ละครั้ง จำนวนนี้จะต้องเป็นจำนวน board lot คือ จำนวนที่หารด้วย 100 ลงตัว โดยทางฝั่งโบรกเกอร์จะต้องทำการตรวจเช็คเอง ถ้าผิดจะให้ใส่ค่าใหม่จนกว่าจะถูกต้อง
- customer name : เก็บรหัสประจำตัวของลูกค้า เป็นหมายเลขจำนวนเต็มจากจอภาพ และนำไปตรวจสอบกับข้อมูลในไฟล์ customer.dat ซึ่งเก็บรายชื่อของลูกค้า
- local_host<HOSTNAME_LEN> เป็นชื่อของ host ที่โบรกเกอร์ส่งข้อมูลไป เพื่อประโยชน์ เมื่อส่งผลลัพธ์ local_host เก็บข้อมูลนี้โดยการใช้คำสั่งของระบบ hostname และเก็บเป็นข้อมูลชนิดสตริง

- nowtime<TIME_STR_LEN> เป็นข้อมูลชนิดสตริง จะเก็บเวลาที่โบรกเกอร์ส่งคำสั่งนั้นๆ เข้าสู่ตลาดหลักทรัพย์ เพื่อทำการซื้อขาย วิธีการเก็บเวลาโดยการเขียนฟังก์ชั่นใน การอ่านเวลาจากระบบ แล้วเก็บเวลานั้นในรูปแบบของสตริง เช่น

  Fri, Mar 17 '95 09:50:03 PM

  Fri : อักษร 3 ตัวแรกแสดงวัน

  Mar : อักษร 3 ตัวแรกแสดงเดือน

  17 : วันที่

  '95 : ปี (ค.ศ)

  09 : ชั่วโมง

  50 : นาที

  03 : วินาที

  PM : เวลาในช่วง 12 ชั่วโมงแรกแทนด้วย AM ,เวลาในช่วง12 ชั่วโมงหลังแทนด้วย PM

- req_id ข้อมูลชนิดค่าจำนวนเต็ม จะแทนหมายเลขของคำสั่งเสนอ ซื้อ-ขาย ในช่วงวันหนึ่ง วิธีการหาข้อมูลนี้จะให้ค่าของคำสั่งแรกที่เข้ามาเป็นหมายเลข 1 และ หมายเลขสำหรับคำสั่งนี้ จะเพิ่มขึ้นครั้งละ 1 เรื่อย ๆ เมื่อมีคำสั่งเข้ามาอีก

ค่าข้อมูลบางส่วนจากที่กล่าวมานี้ บรรจุลงใน struct sendord_type ซึ่งข้อมูลที่นำลงจะเป็นข้อมูลที่เป็นประโยชน์สำหรับฝั่งตลาดหลักทรัพย์ ส่วนข้อมูลอื่นที่เหลือ จะเก็บไว้สำหรับรายงานผล และเป็นหลักฐานในการสั่งซื้อขาย ในไฟล์ order.dat โดยมีฟิลด์ดังนี้

req_id, stockname, order->value, order->price, customer_name , nowtime

ข้อมูลที่จะส่งไปตลาดหลักทรัพย์ จะเก็บลงในรูปแบบของโครงสร้างดังนี้

```
typedef struct sendord_type
                    { int req_id;
                      int stock_number;
                      int id_offer;
                      int price;
                      int value;
                      string local_host<HOSTNAME_LEN>;  } order_type;
```

ข้อมูลที่ใช้ประมวลผลทางด้านตลาดหลักทรัพย์ เมื่อโบรกเกอร์เรียกฟังก์ชั่นการตัดราคาที่ฝั่งตลาดหลักทรัพย์ จะส่งพารามิเตอร์เป็นข้อมูลการซื้อขายชนิด order_type ทางโปรเซสตลาดหลักทรัพย์จะถ่ายข้อมูลเข้าสู่โครงสร้างข้อมูลสำหรับส่งเป็นเมสเซส ไปยังโปรเซส match ซึ่งจะมีการประมวลผลจริง เมสเซสที่จะส่งมีโครงสร้างดังนี้

```
typedef struct my_msgbuf
```

```
{ long  mtype;
  order_type stock_data } MESSAGE;
```

- ตัวแปร mtype เป็นตัวแปรที่แสดงชนิด ของเมสเซสที่จะส่ง จะแบ่งตามลักษณะของฟังก์ชั่นที่เรียกมา

โครงสร้างข้อมูลสำหรับโปรเซส match

```
typedef struct node_tag
        { order_type info;
          struct node_tag *next ;}  NODE_TYPE;
```

เป็นโครงสร้างข้อมูลชนิด NODE_TYPE สำหรับแสดงกลุ่มข้อมูลที่จะบรรจุลงในคิว (queue)เรียกกลุ่มข้อมูลที่มีโครงสร้างแบบนี้ว่า โหนด จะใช้เก็บข้อมูลคำสั่งซื้อ-ขายหุ้น ที่ส่งเข้ามาประมวลผล ใน 1โหนด จะเก็บคำสั่ง 1 คำสั่ง ประกอบด้วยข้อมูลของคำสั่ง และ พอยน์เตอร์ ที่ทำหน้าที่เชื่อมต่อโหนดถัดไป พิจารณารูปที่ 3.3.2 -1 แสดงภาพโครงสร้างของโหนดคำสั่งเสนอซื้อ-ขาย

รูปที่ 3.3.2 -1 แสดงรูปแบบการเก็บคำสั่งซื้อขาย ในโหนด

```
typedef struct queue_tag
        { NODE_TYPE *front;
          NODE_TYPE *rear; } QUEUE_TYPE;
```

เป็นโครงสร้างข้อมูลชนิด QUEUE_TYPE จะเก็บพอยน์เตอร์ที่ใช้สำหรับชี้ตำแหน่งของโหนด ดังนี้

- front จะชี้โหนดที่อยู่ในตำแหน่งแรกของคิว
- rear จะชี้โหนดที่อยู่ในตำแหน่งสุดท้ายของคิว

```
typedef struct stock_tag
        { QUEUE_TYPE *bptr;
          QUEUE_TYPE *optr;
          unsigned long int total_match_num;
          unsigned long int total_match_price;
          int last_match_price; }  STOCK_TYPE;
```

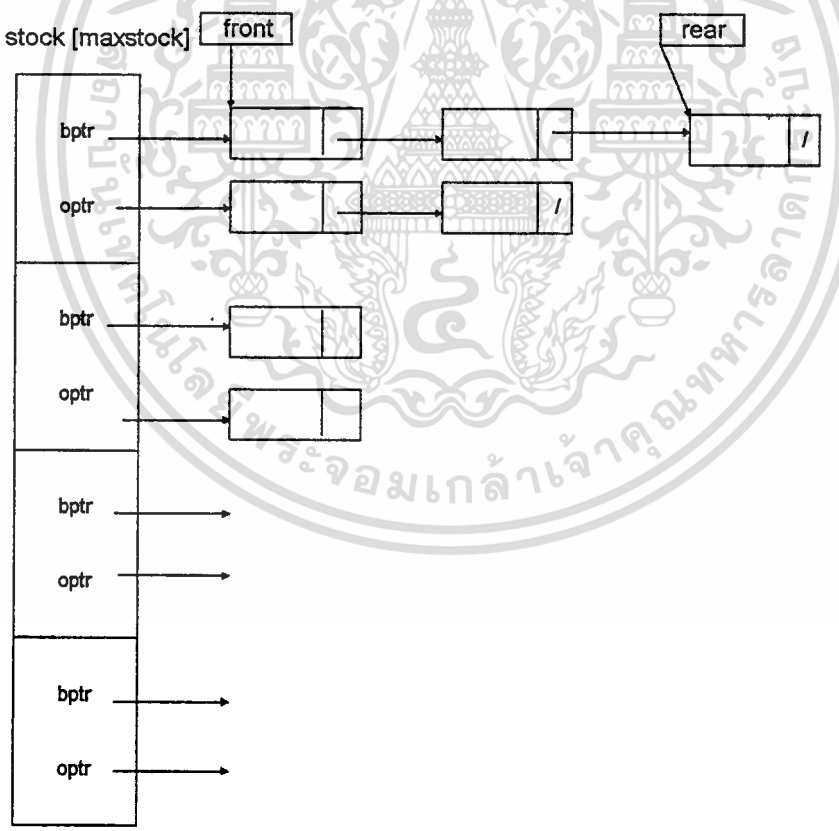
ข้อมูลชนิด STOCK_TYPE เป็นโครงสร้างข้อมูลที่เก็บข้อมูลเฉพาะหุ้นแต่ละตัว ประกอบด้วย

- bptr : เป็นพอยน์เตอร์ที่อ้างถึงคิวของคำสั่งเสนอซื้อของหุ้นตัวนั้น
- optr : เป็นพอยน์เตอร์ที่อ้างถึงคิวของคำสั่งเสนอขายของหุ้นตัวนั้น

- total_match_num : เก็บจำนวนที่มีการซื้อ-ขาย ของหุ้น ภายในวันนั้น
- total_match_price : เก็บมูลค่ารวมที่มีการซื้อขายได้ของหุ้นตัวนั้น
- last_match_price : เป็นราคาที่มีการซื้อขายครั้งล่าสุด

ข้อมูลที่ใช้สำหรับการส่งเมสเซส ระหว่าง โปรเซสที่รับพารามิเตอร์มาจาก RPC แล้วส่งคำสั่งเข้าสู่คิว และโปรเซสที่รับข้อมูลจากคิว นำไปประมวลผล

- MKEY1 : เป็นชื่อของคิวที่ใช้สำหรับการสร้าง ค่าที่ชี้ไปยังเมสเซสคิว ใช้สำหรับสร้างหมายเลขเมสเซสของการส่งคำสั่งซื้อขายหุ้น และคำสั่งขอทราบราคาที่รอการซื้อขาย เนื่องจากทั้งสองฟังก์ชั่นนี้ใช้พื้นข้อมูลเดียวกัน จึงส่งให้อยู่ในคิวที่รอการทำงานเดียวกัน ค่าที่กำหนดเป็น เลขจำนวนเต็ม (int)
- MKEY2 : เป็นชื่อคิวที่ใช้สำหรับสร้างหมายเลขคิว สำหรับส่งผลการตรวจสอบราคาที่รอการซื้อขาย และสถิติของการซื้อขายหุ้นกลับไปยังโปรเซสที่มีคำสั่งขอมา ค่าที่กำหนดมีขนาดเป็น เลขจำนวนเต็ม (int)
- mesg_id1 : เป็นค่าที่ชี้ไปยังเมสเซสคิว ที่ชื่อ MKEY1
- mesg_id2: เป็นค่าที่ชี้ไปยังเมสเซสคิว ที่ชื่อ MKEY2
- PERMS : เป็นเลขจำนวนเต็ม ทำหน้าที่เป็นแฟลก กำหนดขอบเขตการอนุญาตของคิวใหม่ เช่นถ้าให้ PERMS = 0777 หมายความว่า อนุญาตให้อ่าน เขียน และ เรียกทำงานได้ โดยผู้สร้าง กลุ่ม และ ผู้ใช้อื่น



รูปที่ 3.3.2 -2 แสดงรูปแบบการเก็บข้อมูลเฉพาะของแต่ละหุ้น

จากรูปที่ 3.3.2 -2 จะเป็นลักษณะของการเข้าถึงข้อมูลหุ้นโดยใช้วิธี Hash index ในการอ้างถึงข้อมูลหุ้นแต่ละตัว โดยใช้ตัวแปร Stock[MAX_STOCK] เป็นตัวแปรอาเรย์ชนิด STOCK_TYPE MAX_STOCK จะกำหนดจำนวนของหุ้นทั้งหมดที่เปิดให้มีการซื้อขาย เราสามารถอ้างถึงข้อมูลในโหนดได้ดังนี้

1. ข้อมูลที่ต้องการหาเป็นหุ้นหมายเลขอะไร สมมุติว่าเป็นหุ้นหมายเลข 5 เราได้เป็น Stock[5]
2. ข้อมูลนั้นเป็นคำสั่งเสนอซื้อหรือ เสนอขาย

   เสนอซื้อ อ้างโดย Stock[5].bptr

   เสนอขาย อ้างโดย Stock[5].optr
3. อ้างถึงโหนดแรกของคิวคำสั่งซื้อ เช่น Stock[5].bptr->front

   อ้างถึงโหนดสุดท้ายของคิวคำสั่งขาย เช่น Stock[5].optr->rear
4. สมมุติว่าจะเข้าถึงข้อมูลของราคาในโหนดแรก ของคำสั่งซื้อ เช่น Stock[5].bptr->front->info.price;

โครงสร้างข้อมูลที่ใช้ในการส่ง ผลการซื้อขายกลับ มีลักษณะดังนี้

```
typedef struct result_type
                { int req_id;
                  int rest_num;
                  string match_time<TIME_STR_LEN ;} RET_TYPE;
```

โครงสร้างชนิด RET_TYPE ประกอบด้วยข้อมูลย่อยดังนี้คือ

- req_id : เป็นหมายเลขของคำสั่งซื้อขายที่ส่งเข้ามาทำการประมวลผล
- rest_num : จำนวนหุ้นที่ยังเหลืออยู่ในคิวของคำสั่งนั้น
- match_time : เวลาที่สามารถทำการซื้อขาย ได้สำเร็จ ได้มาจากการเก็บเวลา ของระบบขณะนั้น และนำมาแปลงให้อยู่ในรูปของสตริง

3.3.3 ข้อมูลการขอดูโปรเซสการตรวจสอบราคาซื้อ-ขาย

ในการซื้อขายหุ้นนั้น ข้อมูลที่สำคัญต่อการตัดสินใจ ซื้อหรือขายที่ตลาดหลักทรัพย์ส่งมาให้นักลงทุน ใช้ประกอบการพิจารณา ซื้อ-ขาย มีดังนี้

- ราคาซื้อที่ดีที่สุด 3 อันดับ หาได้จากคำสั่งซื้อที่ให้ราคาสูงสุด 3 อันดับ แรกที่อยู่ในคิว
- ราคาขายที่ดีที่สุด 3 อันดับ หาได้จากคำสั่งขายในคิว ที่มีราคาต่ำสุด 3 อันดับแรก
- จำนวนของหุ้นที่ได้มีการซื้อขายกันสำเร็จไปแล้ว ข้อมูลนี้จะเป็นข้อมูลของการซื้อขายขณะใด ๆ ภายใน 1 วัน
- มูลค่ารวมทั้งหมด ที่มีการซื้อขาย หาได้จากค่าสะสมของผลคูณระหว่างจำนวนซื้อขายและราคาซื้อขาย ที่มีการสามารถจับคู่ราคาที่ตรงกันได้สำเร็จ
- ราคาสุดท้ายทีมีการตกลงซื้อขายกันได้

จะเก็บข้อมูลเหล่านี้ อยู่ในรูปแบบของ โครงสร้างข้อมูลดังนี้

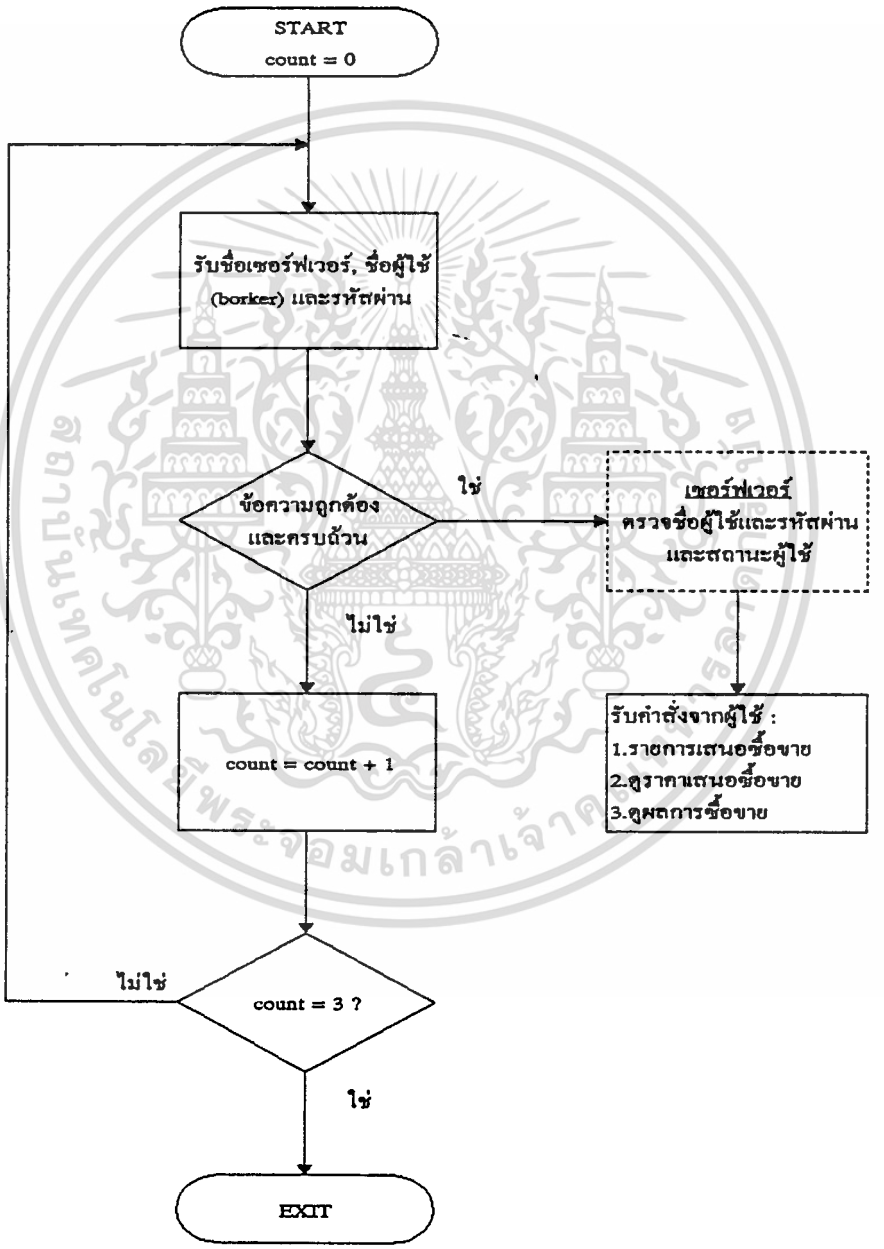
```
typedef struct look_queue_price
                    { int bid1;
                      int bid2;
                      int bid3;
                      int offer1;
                      int offer2;
                      int offer3;
                      unsigned long  total_match_num;
                      unsigned long  total_match_price;
                      int  last_match_price; } PRICEINFO_TYPE;
```

โบรกเกอร์จะทำการเรียกฟังก์ชั่น view_price_1 เพื่อขอดูราคา โบรกเกอร์จะส่งหมายเลขหุ้น ของหุ้นตัวนั้นไปให้ ข้อมูลที่ทางฝั่งโบรกเกอร์จะได้รับกลับ จะเป็นแอดเดรสของโครงสร้างข้อมูล แบบ PRICEINFO_TYPE

บทที่ 4

อัลกอริทึม

4.1 ขั้นตอนของอัลกอริทึมของการขอเข้าระบบ



รูปที่ 4.1-1 แสดงการรับคำสั่งในการขอเข้าระบบทางฝั่งโบรกเกอร์

จากรูป 4.1 -1 เป็นฟังก์ชัน Logon ซึ่งเป็นฟังก์ชันแรกที่ต้องใช้งาน เพื่อร้องขอให้แอพพลิเคชันอนุญาตให้ทำงานในฟังก์ชันอื่นต่อไป และติดต่อกับตลาดหลักทรัพย์ เพื่อซื้อขายหลักทรัพย์ หรือขอข้อมูลจากตลาด เป็นต้นว่า ราคาหลักทรัพย์, ปริมาณการซื้อขายหลักทรัพย์ และมูลค่าการซื้อขายหลักทรัพย์ปัจจุบัน การทำงานของฟังก์ชันสามารถอธิบายโดยชาร์ทได้ดังนี้

1. โปรแกรมรับชื่อของโฮสต์หรือ เซอร์ฟเวอร์ของตลาด (โครงงานนี้อนุญาตให้ผู้ใช้ป้อนชื่อโฮสต์ได้ เพื่อให้เห็นว่ามีการติดต่อกับโฮสต์อื่นได้จริง) รับชื่อผู้ใช้ซึ่งคือโบรกเกอร์ และรับรหัสผ่านของผู้ใช้

2. ตรวจสอบข้อความที่ผู้ใช้ป้อนในขั้นตอนแรก ว่าครบถ้วนหรือไม่

- ถ้าข้อความครบถ้วน โปรแกรมไคลเอ็นต์จะเรียกใช้ฟังก์ชันไปยังโฮสต์ที่ผู้ใช้ระบุ โดยมีชื่อผู้ใช้และรหัสผ่านเป็นอาร์กิวเมนต์ เพื่อให้ฟังก์ชันในเซอร์ฟเวอร์ตรวจความถูกต้องตามเงื่อนไขต่าง ๆ ที่กำหนดไว้ รายละเอียดจะกล่าวในส่วนอัลกอริทึมของฟังก์ชันบนเซอร์ฟเวอร์

- ถ้าข้อความไม่ครบถ้วน ให้เพิ่มค่า count อีกหนึ่ง เพื่อนับจำนวนครั้งที่ผู้ใช้พยายาม log เข้ามาในระบบ แล้วตรวจค่า count ว่าเท่ากับ 3 หรือยัง ถ้า count มีค่าเท่ากับ 3 ให้สิ้นสุดการทำงาน ถ้า count ไม่เท่ากับ 3 จะทำงานรับชื่อโฮสต์ ชื่อผู้ใช้ และรหัสผ่านอีกครั้ง

กรณีที่ข้อความครบถ้วน และเซอร์ฟเวอร์ตรวจสอบแล้วว่าอนุญาตให้ผู้ใช้ทำงานในฟังก์ชันต่อไปได้ โปรแกรมก็จะเข้าสู่สภาวะรอรับคำสั่งจากผู้ใช้ต่อไป คำสั่งที่รอรับจากผู้ใช้ประกอบด้วย คำสั่งสร้างรายการ (order) เสนอซื้อขาย, คำสั่งขอดูราคาเสนอซื้อขายของหลักทรัพย์ที่ต้องการ และ ดูผลการซื้อขายหลักทรัพย์ของรายการซื้อขายที่เคยสร้าง

ส่วนขั้นตอนการทำงานของตลาดหลักทรัพย์ในการตรวจสอบความถูกต้องของข้อมูลรหัสผ่านเป็นดังรูปที่ 4.1 -2

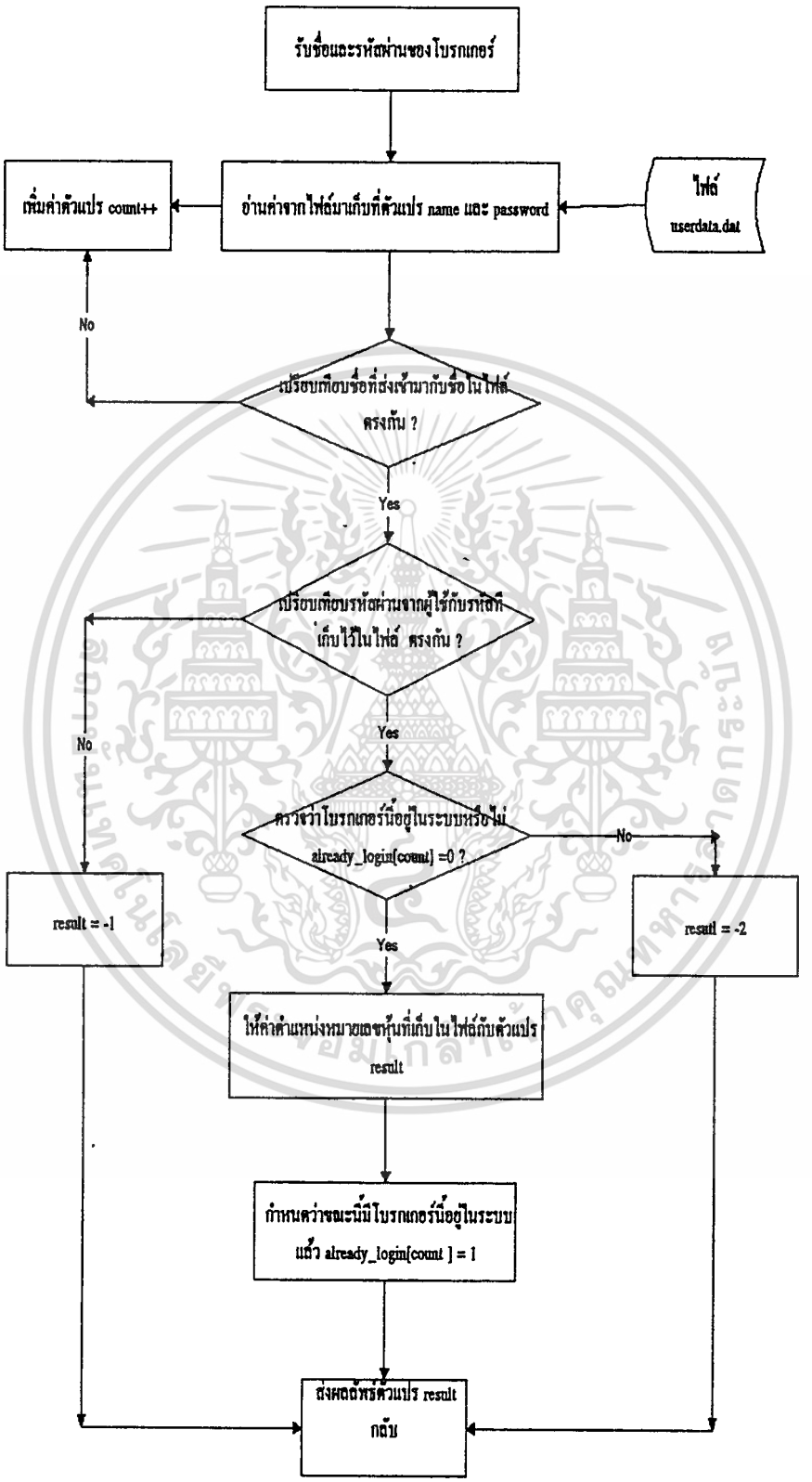
โพรเซสทางตลาดหลักทรัพย์เป็นโพรเซสที่ทำงานอยู่เบื้องหลัง และรอการเรียกฟังก์ชั่นขอเข้าระบบ เมื่อมีโบรกเกอร์ใดต้องการเข้าร่วมการซื้อขายก็จะส่ง ชื่อและ รหัสผ่านเข้ามา ทางตลาดหลักทรัพย์จะเปิดไฟล์เปรียบเทียบว่าชื่อและรหัสผ่านตรงกันหรือไม่

4.2 ขั้นตอนการทำงานของคำสั่งซื้อขายหลักทรัพย์

หลังจากผู้ใช้ log เข้ามาในระบบ และผู้ใช้ (โบรกเกอร์) ต้องการสร้างรายการ (order) การซื้อขาย จึงต้องเข้ามาทำงานในฟังก์ชันนี้

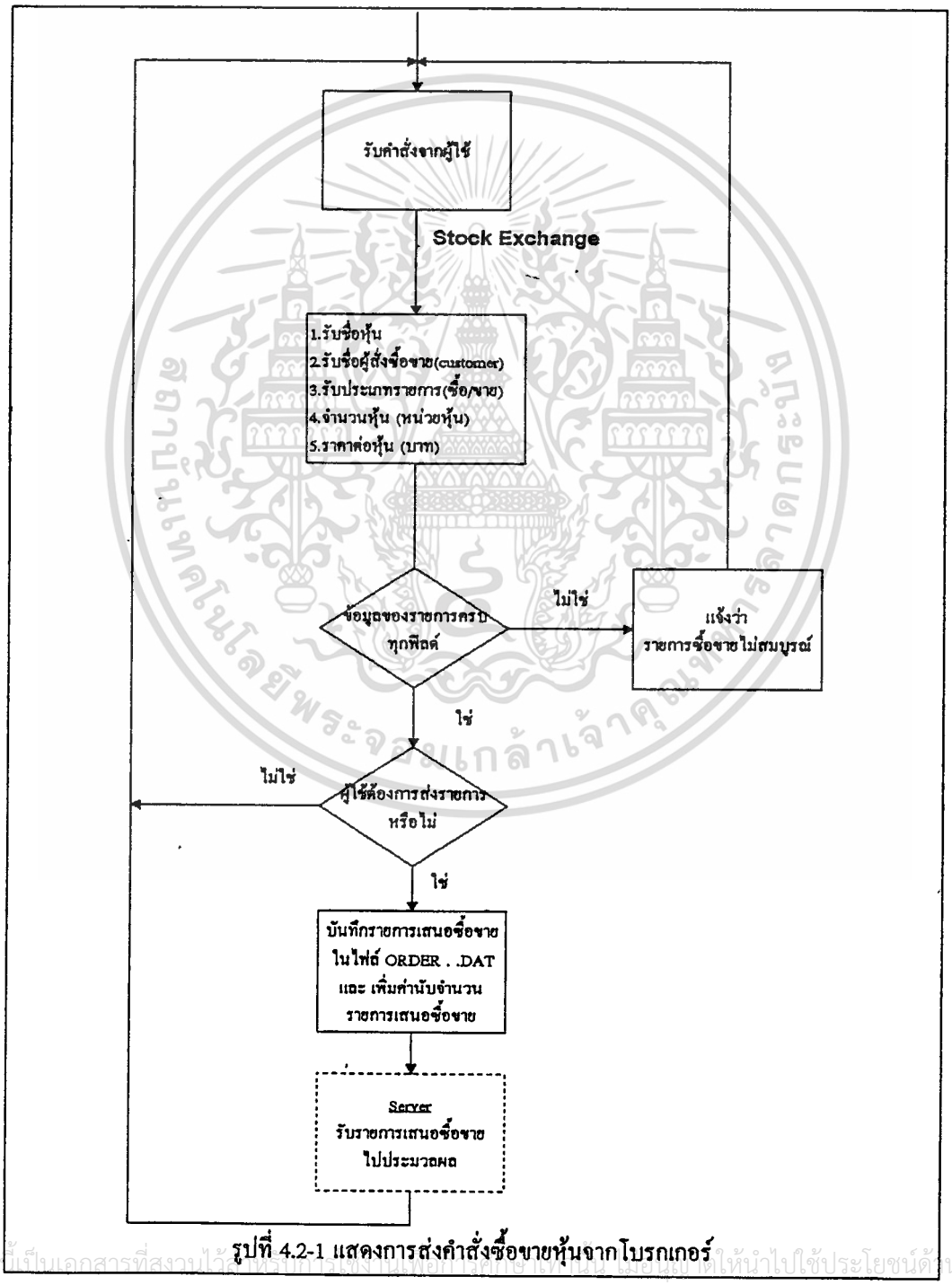
โปรแกรมต้องการรายละเอียดของรายการเสนอซื้อขาย อันประกอบด้วย ชื่อหุ้น, ชื่อผู้สั่งซื้อขาย (customer), ประเภทของรายการที่เสนอ (ซื้อหรือขาย, bid หรือ offer), จำนวนหุ้น (หน่วยเป็นหุ้น) และ ราคาต่อหุ้น (บาท) เมื่อรับรายละเอียดต่าง ๆ เสร็จ ต้องตรวจว่าให้ค่าครบทุกประการหรือไม่

- ถ้ารับรายละเอียดได้ไม่ครบ จะแสดงข้อความบอกแก่ผู้ใช้ว่า รายการซื้อขายไม่สมบูรณ์ จะยกเลิกการส่งรายการนั้นทันที แล้วกลับไปรอรับคำสั่งจากผู้ใช้ต่อไป



รูปที่ 4.1 -2 แสดงการทำคำสั่งขอเข้าระบบโดยตลาดหลักทรัพย์

- ถ้าได้รับรายละเอียดครบ จะถามผู้ใช้ว่าต้องการส่งรายการนั้นไปยังตลาดหลักทรัพย์หรือไม่ ถ้าตอบ “y” คือให้ส่งรายการ ฟังก์ชันจะบันทึกรายการเสนอซื้อขายในแฟ้มข้อมูลชื่อ ORDER.DAT และเพิ่มค่าตัวแปรสำหรับนับจำนวนรายการที่เสนอซื้อขาย แล้วส่งรายการนั้นไปยังตลาด (เซอร์ฟเวอร์) เพื่อประมวลผล เมื่อการส่งสมบูรณ์แล้ว ฟังก์ชันจะรีเทิร์นกลับมารอรับคำสั่งอีกครั้งทันที โดยไม่ต้องรอค่าที่รีเทิร์นกลับมาจากฟังก์ชันของเซอร์ฟเวอร์ แต่ถ้าผู้ใช้ปฏิเสธ การส่งโดยตอบ “n” ฟังก์ชันจะกลับไปรอรับคำสั่งต่อไปโดยไม่บันทึกรายการซื้อขายนั้นลงแฟ้มข้อมูล พิจารณารูปที่ 4.2-1 ประกอบ



รูปที่ 4.2-1 แสดงการส่งคำสั่งซื้อขายหุ้นจากโบรกเกอร์

ตามปกติแล้วลำดับในคิวของหุ้นตัวใดตัวหนึ่งจะเรียงตามราคาดังนี้

ในคิวของรายการเสนอซื้อ จะเรียงจากราคาแพงกว่าไปยังราคาต่ำกว่า ถ้าราคาเท่ากันให้เรียงตามลำดับก่อนหลัง

ในคิวของรายการเสนอซื้อ จะเรียงจากราคาถูกกว่าไปยังราคาสูงกว่า กรณีราคาเท่ากันจะให้เรียงตามลำดับก่อนหลังเช่นกัน ดังนั้นในโครงงานนี้จึงมี หลักเกณฑ์ในการจับคู่ราคาซื้อขาย หุ้น ดังนี้

1. จับคู่ตัดราคาซื้อขายภายในหุ้นที่มีชื่อเดียวกัน ที่เก็บอยู่ในคิว ซึ่งโหนดในคิวนั้นเป็นคำสั่งที่ไม่สามารถตัดราคากันเองได้ เมื่อมีคำสั่งเข้ามาใหม่ ก็จะเปรียบเทียบกับหุ้นที่อยู่ในคิวเลย
2. คำสั่งที่เสนอเข้ามา จะต้องพิจารณาว่าเป็นคำสั่งเสนอซื้อ หรือเสนอขาย กรณีที่เป็นคำสั่ง เสนอซื้อ จะนำไปเปรียบเทียบกับข้อมูลที่อยู่ในคิว เสนอขาย

กรณีที่เป็นคำสั่ง เสนอขาย จะนำไปเปรียบเทียบกับข้อมูลที่อยู่ในคิว เสนอซื้อ

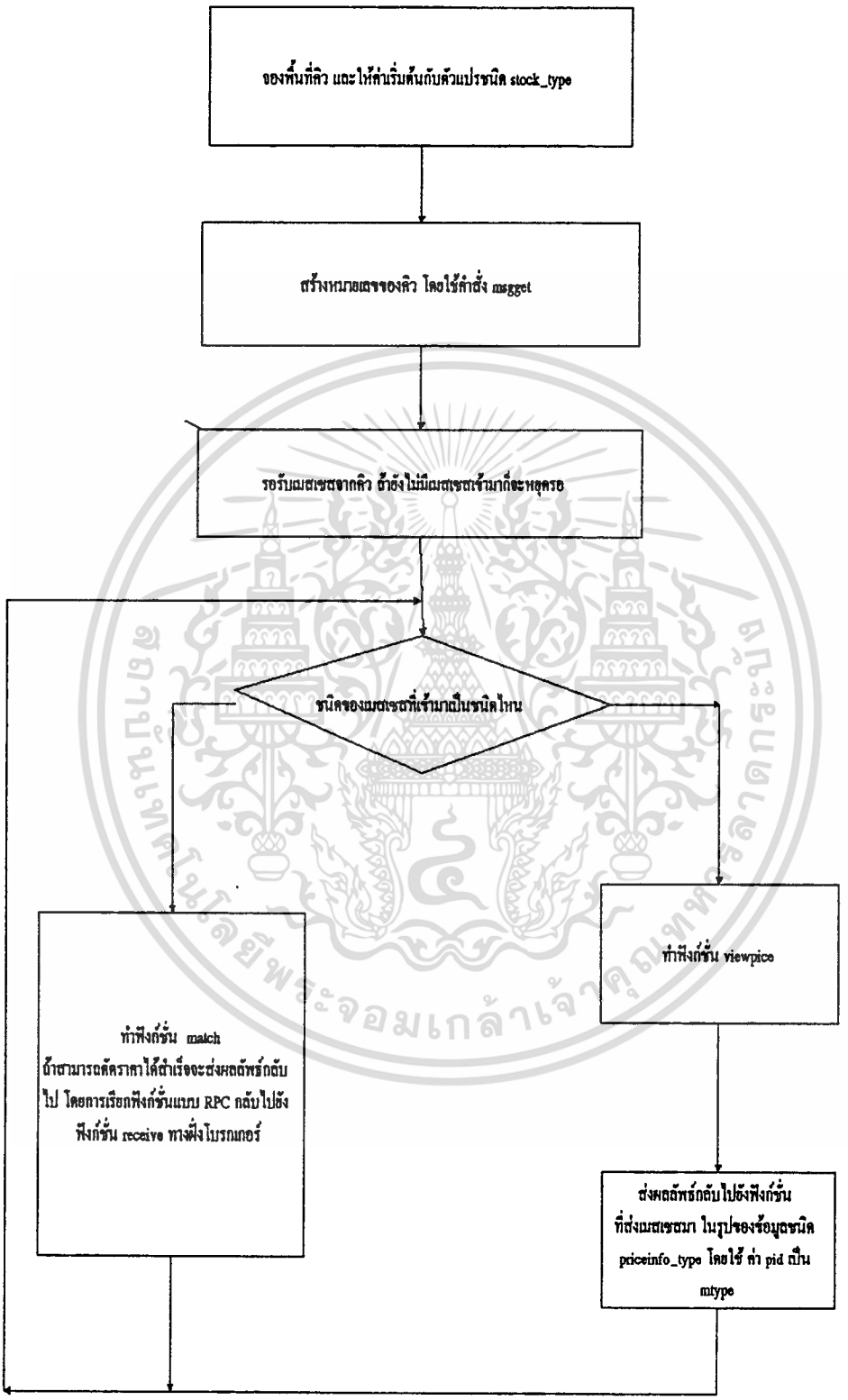
3. จะทำการเปรียบเทียบราคาที่โหนดแรกของคิวเท่านั้น จะไม่สามารถทำคำ

สั่งการจับคู่ราคาหุ้นครอบคลุมถึงการซื้อขาย แบบที่เรียกว่า เคาะซื้อ หรือ เคาะขาย เช่นกรณีที่เสนอราคาซื้อสามารถจับคู่ได้กับราคาขายหลายค่า

สำหรับอัลกอริทึมการจับคู่ราคาทางฝั่งตลาดหลักทรัพย์ จะแบ่งเป็น สองโพรเซสดังที่ได้กล่าวไปแล้ว อัลกอริทึมของโพรเซส servrpc ซึ่งเป็นโพรเซสที่คอยรับคำสั่งจากโบรกเกอร์ผ่านทาง RPC มีดังนี้

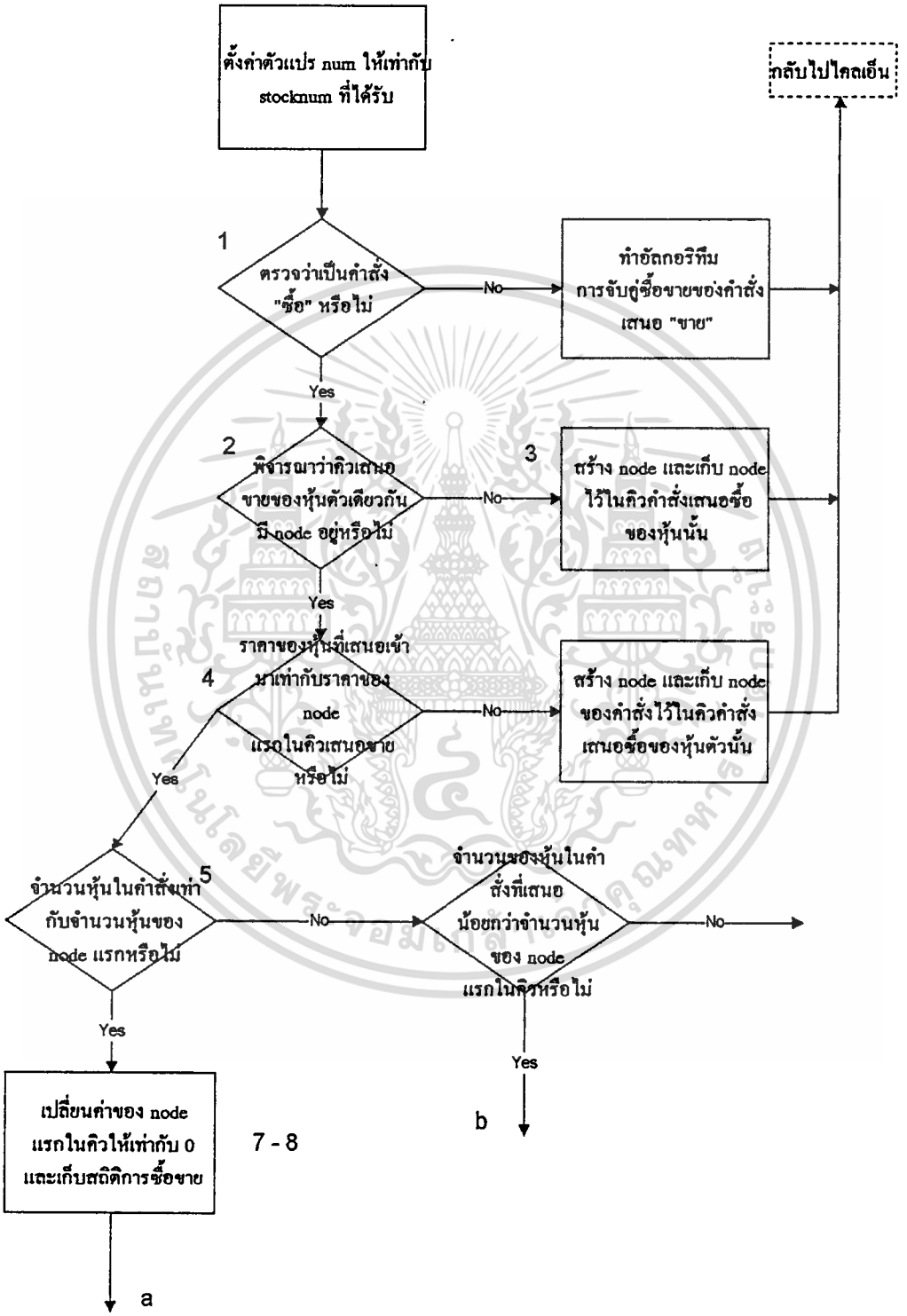
1. รับค่าแอดเดรสของข้อมูลโครงสร้างชนิด order_type แล้วนำไปถ่ายให้กับตัวแปรในโครงสร้างข้อมูลแบบเมสเซส สำหรับส่งข้อมูลผ่านเมสเสจคิว
2. ใส่ชนิดของข้อมูลของเมสเสจที่จะส่งในตัวแปร mtype
3. ทำการหาหมายเลขของเมสเสจที่จะส่งจากคำสั่ง msgget โดยจะป้อน MKEY และ PERMS ระบุชนิดและขอบเขตของเมสเสจที่จะส่งนั่น
4. ทำการส่งเมสเสจ โดยจะส่งแอดเดรสของตัวแปรแบบ โครงสร้างเมสเสจ ไปโดยคำสั่ง msgsnd ในรูปแบบของการติดต่อแบบ เมสเสจคิว ไปยังฟังก์ชั่นที่ทำหน้าที่ในการตัด

พิจารณาที่โพรเซส match จะรอรับเมสเจในการส่งคำสั่งมา ซึ่งในคิวจะประกอบด้วยเมสเสจ 2 ชนิด คือ คำสั่งเสนอซื้อ ขาย และคำสั่งในการขอดูราคาหุ้น เหตุที่นำคำสั่งทั้งสองมาอยู่ในคิวเดียวกันเนื่องจาก ต้องการใช้ข้อมูลส่วนกลาง เดียวกัน ทางตลาดหลักทรัพย์จะทำงานที่ละคำสั่งที่เข้ามา ตามลำดับก่อนหลัง พิจารณารูปของการทำงานในโพรเซส match ในรูปที่ 4.2 -2

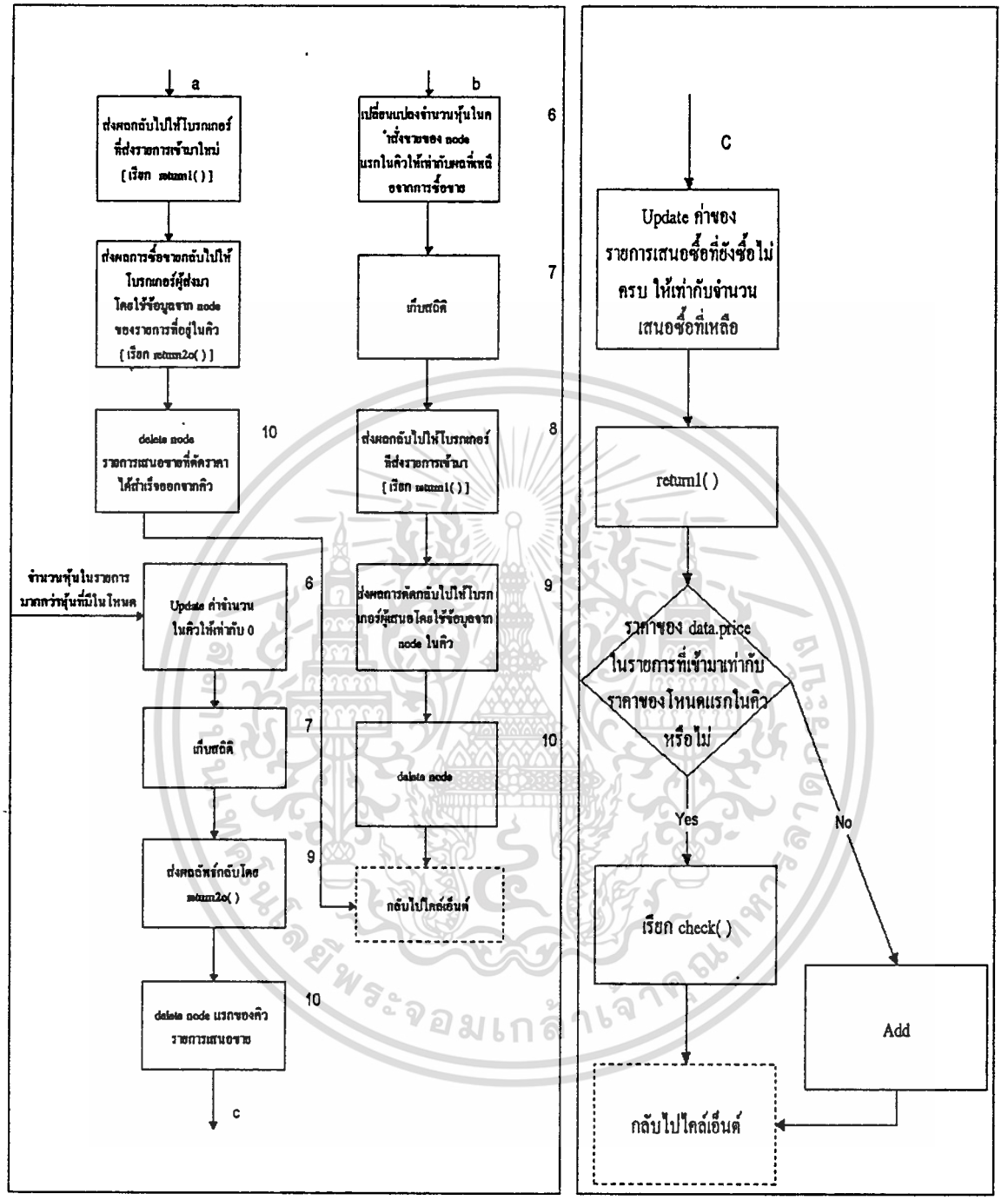


รูปที่ 4.2_2 แสดงโครงสร้างโดยรวม ของโพรเซส match

ในโพรเซส match จะมีการแยกทำงานฟังก์ชัน ซึ่งพิจารณาตาม ชนิดของเมสเสจที่ส่งมา สำหรับอัลกอริทึมในการตัดราคาซื้อ ขาย หุ้นมีดังนี้



รูปที่ 4.2_3.1 แสดง ขั้นตอนการตัดราคาซื้อ ขายหุ้น



รูปที่ 4.2-3.2 แสดงขั้นตอนการตัดราคาซื้อขายหุ้น (ต่อ) (ภาพซ้าย)

รูปที่ 4.2-3.3 แสดงขั้นตอนการตัดราคาซื้อขายหุ้น (ต่อ) (ภาพขวา)

คำอธิบายเพิ่มเติมจากโฟลว์ชาร์ต

1. พิจารณาคำสั่งที่เข้ามา เพื่อแยกกรณีในการจับคู่ราคาหุ้น ว่าเป็นคำสั่งซื้อ หรือคำสั่งขาย
   กรณีที่เป็นคำสั่งซื้อ อัลกอริทึมก็จะเป็นการตัดราคากับโหนดในคิว คำสั่งขาย และในทางตรงกันข้าม ถ้าคำสั่งที่เข้ามาเป็นคำสั่งขาย ก็จะทำอัลกอริทึม ในการตัดราคากับโหนดในคิวคำสั่งซื้อ ส่วนขั้นตอนอื่น ๆ ในกระบวนี้

   การจับคู่ราคา ก็จะคล้าย ๆ กัน ในที่นี้จะแสดงอัลกอริทึมของคำสั่งที่เพิ่งเข้ามาเป็นคำสั่งซื้อเท่านั้น
2. ตรวจว่าที่หุ้นตัวที่เราจะสั่งซื้อนั้นมีผู้ที่ต้องการขาย เสนอขายเข้ามาหรือไม่ ถ้ายังไม่มีผู้ใดเสนอเข้ามาเลย จะต้องเก็บคำสั่งนั้นไว้ในคิวคำสั่งเสนอซื้อ เพื่อรอคอยผู้ที่ต้องการขายในราคาที่ตรงกัน
3. การเก็บข้อมูลเข้าไว้ในคิว จะต้องนำข้อมูลที่ส่งเข้ามาไปสร้างเป็นโหนด ก่อน ซึ่งโหนดนั้น จะประกอบด้วยส่วนที่เป็นข้อมูลและส่วนที่เป็นพอยน์เตอร์ การอ้างถึงโหนดใด ๆ จะอ้างโดยพอยน์เตอร์ที่ชี้โหนดนั้น พิจารณาฟังก์ชั่นที่สร้างโหนด

   NODE_TYPE *Make_Node(item)

   order_type item;

เมื่อเราให้ค่าส่วนที่เป็นข้อมูลและ กำหนดให้ส่วนที่เป็นพอยน์เตอร์เป็น NULL แล้ว ก็จะส่ง ตัวแปรพอยน์เตอร์ที่ชี้โหนดนั้นเป็นค่าที่ส่งกลับ ค่านี้จะเข้าไปเป็นพารามิเตอร์สำหรับฟังก์ชั่นในAddnode ซึ่งมีรูปแบบดังนี้

Addnode (node_ptr,queue_ptr)

NODE_TYPE *node_ptr;

QUEUE_TYPE *queue_ptr;

queue_ptr เป็นพารามิเตอร์ชนิดพอยน์เตอร์ของคิว ที่จะนำโหนดเข้ามาเก็บ จากที่ได้กล่าวไปข้างต้นแล้วว่า ลำดับการเรียงโหนดในคิวจะเรียงตามราคาของโหนดดังนี้

คิวของคำสั่งเสนอซื้อ (bid) จะเรียงจากราคาสูงกว่า ไปยัง ราคาต่ำกว่า

คิวของคำสั่งเสนอขาย (offer) จะเรียงจากราคาต่ำกว่า ไปยัง ราคาสูงกว่า

กรณีที่ราคาเท่ากัน การแทรกเข้าคิว จะต้องเรียงตามลำดับก่อนหลัง (first com first serve)

อัลกอริทึมของการบรรจุโหนดเข้าคิว (Addnode)เป็นดังนี้

1. ถ้าพอยน์เตอร์ที่ชี้ โหนด เป็น NULL จะแสดงข้อความผิดพลาดออกมา
2. ถ้าไม่มีโหนดในคิวที่จะบรรจุ ให้ใส่โหนดนั้นเป็นโหนดแรกในคิว และกำหนดให้พอยน์เตอร์ front และพอยน์เตอร์ rear ชี้ที่โหนดเดียวกัน
3. กรณีใส่โหนดที่ท้ายคิว

กรณีที่จะใส่โหนดที่คิวคำสั่ง ซื้อ(bid queue) และข้อมูลราคาของโหนดที่เข้ามา น้อยกว่าหรือเท่ากับ ราคาของโหนดสุดท้าย(ชี้โดยพอยน์เตอร์ rear) ให้ใส่โหนดนั้นที่ตำแหน่งท้ายคิวเลย

กรณีที่จะใส่โหนดที่คิวคำสั่งขาย(offer queue) และราคาของโหนดที่จะใส่มากกว่าหรือเท่ากับ ราคาของโหนดสุดท้ายในคิว ให้ใส่โหนดนั้นที่ตำแหน่งท้ายคิว

4. กรณีใส่โหนดที่ต้นคิว กรณีจะใส่โหนดที่คิวคำสั่งซื้อ และข้อมูลที่เข้ามามีราคา สูงกว่า ราคาของโหนดแรกในคิว ให้แทรกโหนดนี้ไว้ ที่หน้าตำแหน่งที่พอยน์เตอร์ front ชี้อยู่ และเปลี่ยนให้พอยน์เตอร์ frontให้มาชี้ที่โหนดแรกของคิว

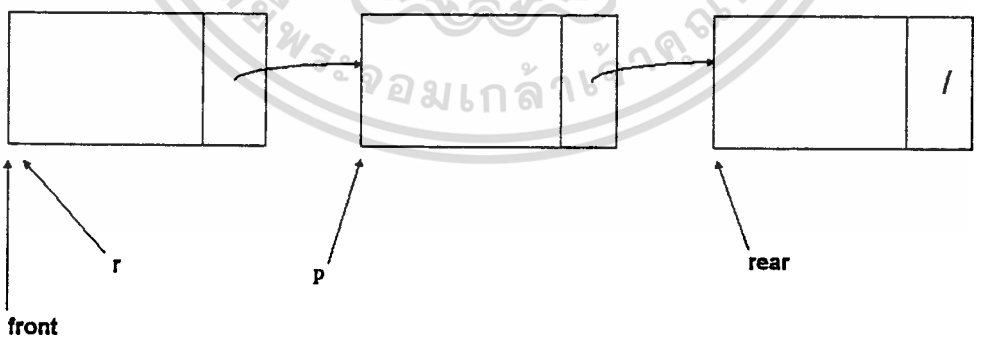
กรณีจะใส่โหนดที่คิวคำสั่งขาย ถ้าข้อมูลที่เข้ามาต่ำกว่า ราคาที่โหนดแรกของคิว ให้แทรกโหนดนี้ไว้ หน้าตำแหน่งที่พอยน์เตอร์ front และเปลี่ยนพอยน์เตอร์ ให้ไปชี้ที่โหนดแรกของคิวใหม่

5. จากกรณีที่ 3 และ 4 ถ้าไม่สามารถใส่ที่ตำแหน่งแรก และตำแหน่งสุดท้ายของคิว แสดงว่าโหนดนี้ควรจะแทรกอยู่ระหว่าง โหนดแรก และ โหนดสุดท้าย วิธีนี้จะต้องเปรียบเทียบข้อมูลแต่ละโหนดจากต้นคิวไปยังท้ายคิว เพื่อหาตำแหน่งที่เหมาะสม

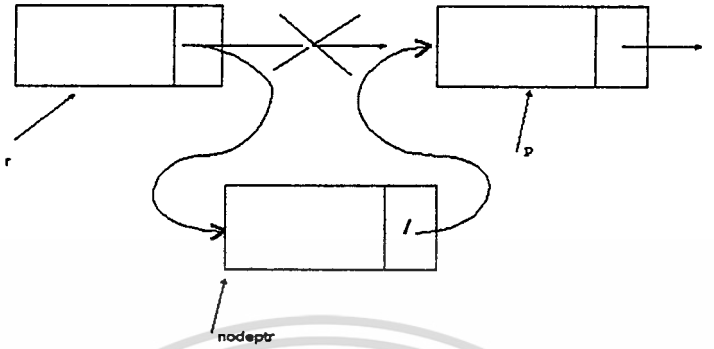
วิธีการเข้าถึงข้อมูลของโหนดในคิว ได้ออกแบบให้มีลักษณะดังนี้

กำหนดตัวแปรพอยน์เตอร์แบบคิว 2 ตัวคือ r และ p โดยพอยน์เตอร์ p จะอยู่หน้าพอยน์เตอร์ r หนึ่งขั้นเสมอ ถ้ามีการเลื่อนพอยน์เตอร์ p พอยน์เตอร์ r ก็จะเลื่อนตาม พิจารณาดังรูป 4.2-4

ถ้าต้องการใส่โหนดที่คิวของคำสั่งขาย ให้เลื่อนพอยน์เตอร์ p และ r ไปข้างหน้าจนกว่าข้อมูลราคาของโหนดที่จะแทรก มากกว่า ราคาของโหนดที่พอยน์เตอร์ p ชี้อยู่ จากนั้นให้แทรกโหนดนั้นเข้าระหว่าง โหนดที่ r และ p ชี้ มีการเปลี่ยนแปลงการชี้ของพอยน์เตอร์ดังรูปที่ 4.2-5



รูปที่ 4.2-4 แสดงการใช้พอยน์เตอร์ช่วยในการค้นหาโหนด



รูปที่ 4.2-5 แสดงวิธีแทรกโหนดในคิว

ในทางกลับกัน ถ้าต้องการใส่โหนดเข้าที่คิวคำสั่ง ซื้อ ให้เลื่อนพอยน์เตอร์ p และ r ไปทางท้ายคิว จนกว่าข้อมูลราคาของโหนดที่ จะแทรกน้อยกว่าข้อมูลราคาของโหนดที่พอยน์เตอร์ p ชี้ และแทรกโหนด ระหว่างพอยน์เตอร์ p และ r

วิธีนี้จะครอบคลุมไปถึงกรณีที่ มีโหนดที่มีราคาเท่ากัน ก็จะใส่โหนดที่เข้ามาใหม่ไว้อยู่ในลำดับหลังของโหนดเก่าที่มีราคาเท่ากันด้วย

6. การเปลี่ยนแปลงจำนวนหุ้นของโหนดที่อยู่ในคิว มีหลักเกณฑ์ดังนี้

กรณีที่ราคาเสนอเข้ามาเท่ากับราคาที่อยู่ในคิว และจำนวนหุ้นของโหนดใหม่ เท่ากับจำนวนหุ้นของโหนดที่อยู่ในคิวพอดี ให้เปลี่ยนแปลงจำนวนหุ้นในคิว ให้เป็น 0 ก่อน เพื่อนำค่านี้ไปใช้ในฟังก์ชั่นการส่งค่ากลับ

กรณีที่ราคาเสนอเท่ากัน และจำนวนที่สั่งใหม่ น้อยกว่าจำนวนของโหนดแรกในคิว ภายหลังจากที่ทำการเปลี่ยนแปลงจำนวนหุ้นที่เหลือให้กับหุ้นที่ยังอยู่ในคิวแล้ว ก็จะกำหนดให้ตัวแปรโกบอล diff มีค่าเป็น 0 เพื่อเป็นประโยชน์ของฟังก์ชั่นส่งค่ากลับ

7.เก็บสถิติการซื้อขายหุ้น เป็นข้อมูลสะสมในหนึ่งวันเท่านั้น ค่าที่เก็บมีดังนี้

1. จำนวนที่ขายได้ทั้งหมดของหุ้น จะเพิ่มทุกครั้งที่มีการซื้อขายสำเร็จ
2. จำนวนมูลค่ารวมการซื้อขายของหุ้นตัวนั้น หาได้จากการบวกเพิ่มของ จำนวนที่ซื้อขายได้ขณะนั้น คูณกับ มูลค่าที่ซื้อขายขณะนั้น กับมูลค่ารวมเดิม
3. ราคาที่มีการซื้อขายได้สำเร็จครั้งล่าสุด

8. ส่งผลลัพธ์ที่ตัดได้ กลับไปยัง โฮสต์ ของผู้ส่งคำสั่งซื้อขาย ที่เพิ่งเข้ามาใหม่ ข้อมูลที่ส่งกลับเป็นข้อมูลในโครงสร้างของ RET_TYPE ประกอบด้วย

timestr : เวลาของระบบขณะที่ทำการซื้อขายหุ้นตัวนั้นสำเร็จ

req_id : หมายเลขของคำสั่งซื้อ ขาย

rest_num : จำนวนหุ้นที่ยังไม่สามารถ ซื้อขายได้ ของคำสั่งนั้น และจะส่งผลลัพธ์กลับ จะใช้รูปแบบของ RPC

11. ในกรณีที่ node ที่เข้ามาใหม่มีจำนวนหุ้นมากกว่าโหนดเก่า หลังจากที่ลบโหนดต้นคิวออกไปแล้ว ให้เปรียบเทียบกับโหนดใหม่ที่โหนดแรกของคิว ถ้ามีราคาเท่ากันให้ทำฟังก์ชั่นในการตัดราคาอีกครั้งหนึ่ง เป็นการเรียกการทำงานแบบ recursive

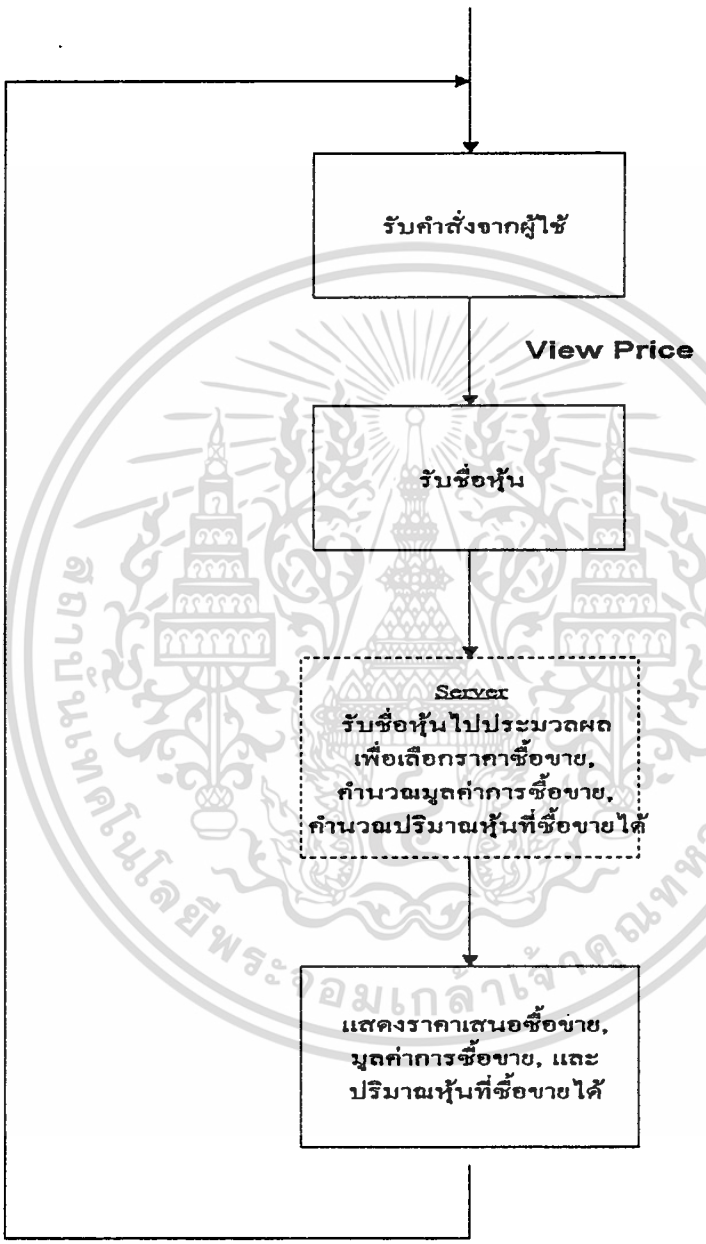
หลังจากที่จัดการกับคำสั่ง ที่เข้ามาใหม่เรียบร้อยแล้ว ระบบก็จะกลับไปรอ รับเมสเซสคำสั่งซื้อขาย หรือคำสั่งขอดูราคาหุ้นที่จะเข้ามาใหม่

4.3 ขั้นตอนการทำงานของฟังก์ชั่น ในการของดูราคาหุ้น

ฟังก์ชัน view_price_1 เป็นฟังก์ชันที่ผู้ใช้ ต้องการดูราคาเสนอซื้อ สูงสุดสามอันดับแรก และ ราคาเสนอขายต่ำสุด สามอันดับแรก และ สถิติ การซื้อขายของหุ้นที่ผู้ใช้เลือก พิจารณาตามรูปดังรูปที่ 4.3-1

ส่วนการทำงานของ ฟังก์ชั่น viewprice ทางตลาดหลักทรัพย์มีขั้นตอนดังนี้

1. รับหมายเลขหุ้นจากโบรกเกอร์ และ เก็บค่าของหมายเลขโปรเซสนั้นด้วย หมายเลขโปรเซสนี้จะใช้ในกรณีที่มีโปรเซสที่สร้างขึ้นมาเรียกฟังก์ชั่น นี้ หลาย ๆโปรเซส จะใช้หมายเลขโปรเซสกำหนดชนิดของเมสเสจที่จะส่งเมื่อทางตลาดหลักทรัพย์ ส่งผลลัพธ์กลับมาทางเมสเสจคิวแล้ว โปรเซสนั้นก็จะรับผลเฉพาะหุ้นที่ตนสั่ง



รูปที่ 4.3-1 แสดงขั้นตอนการเรียกคำสั่งขอดูราคาหุ้น

2. กระบวนการเก็บผลลัพธ์ ทางโปรเซส match จะรับโปรเซสที่มี mtype แสดงความต้องการของดูราคาหุ้นตามที่ได้ตกลงกันไว้ มาทำงานในฟังก์ชั่น นี้

วิธีการก็คือ การเข้าไปตรวจสอบที่คิว ของหมายเลขหุ้นที่ได้ส่งเข้ามาเป็น พารามิเตอร์ และหาราคาที่ต่างกันสาม ราคา จะทำการค้นหาตั้งแต่ต้นคิว ของคิวราคาเสนอซื้อ และคิวราคาเสนอขาย ซึ่งเป็นราคาที่รอคำสั่งซื้อ

หรือขายที่มีราคาตรงกันเข้ามา จากนั้นจะอ่านค่าสถิติการซื้อขาย ของหุ้นตัวนั้น และส่งกลับเป็นตัวแปรโครงสร้างแบบ PRICEINFO_TYPE ดังที่ได้กล่าวแล้วในบทที่ 3

ฟังก์ชัน view_result

ทำงานเมื่อผู้ใช้ต้องการดูผลการซื้อขาย ของรายการเสนอซื้อขาย ที่มี หมายเลขหุ้นตามที่ผู้ใช้ป้อน ฟังก์ชันนี้จะมีการทำงานที่โบรกเกอร์เท่านั้น โดยใช้วิธีการอ่านไฟล์ที่เก็บผลการซื้อขายไว้ พร้อมกับแสดงผลให้ผู้ใช้ทราบ มีขั้นตอนการทำงานตามรูปที่ 4.3-2 ดังนี้

ฟังก์ชัน View Result ทำงานเมื่อผู้ใช้ต้องการซื้อขาย ของรายการเสนอซื้อขายที่มี Request ID ตามที่ผู้ใช้ป้อน ฟังก์ชันเริ่มรับ Request ID จากผู้ใช้ แล้วนำหมายเลข Request ID มาเปรียบเทียบกับค่าตัวแปรสำหรับนับจำนวนรายการซื้อขายในขณะปัจจุบัน

- ถ้าค่า Request ID มากกว่าจำนวนรายการ จะกลับไปรอรับค่า Request ID ใหม่ แต่ถ้าผู้ใช้กดคีย์เอ็นเทอร์โดยไม่ใส่ค่าใด ๆ ฟังก์ชันจะกลับออกไปรับคำสั่งจากผู้ใช้ทันที

- ถ้าค่า Request ID น้อยกว่าหรือเท่ากับจำนวนรายการ จะเปิดไฟล์ ORDER.DAT และ RESULT.DA

เพื่ออ่านข้อมูลของ Request ID นั้น แล้วนำมาแสดงผลในหน้าต่าง ข้อมูลที่แสดงนั้นประกอบด้วย

หมายเลข Request ID, ชื่อผู้เป็นเจ้าของรายการ, ชื่อหุ้นในรายการ, จำนวนหุ้นที่เสนอซื้อหรือขาย, จำนวนหุ้นที่สามารถตกลงซื้อหรือขายได้, จำนวนหุ้นที่คงเหลือจากการซื้อหรือขาย, ราคาต่อหุ้น, มูลค่าของการตกลงซื้อขายหุ้นในรายการนั้น (บาท), เวลาที่สร้างรายการ และ เวลาการตกลงซื้อขายครั้งล่าสุด

ฟังก์ชัน Save Result

เป็นโพรเซสที่ทำงานอยู่เบื้องหลังของโปรแกรมของโบรกเกอร์ (ทำงานอยู่ในฝั่งไคล์เอ็นต์) มีหน้าที่บันทึกผลการตกลงซื้อขายลงในแฟ้มข้อมูลชื่อ RESULT.DAT ของผู้ใช้แต่ละราย ข้อมูลที่โพรเซสนี้บันทึกได้มาจากโพรเซสที่ทำงานอยู่ในฝั่งเซอร์ฟเวอร์ (หรือในตลาด)

เมื่อการตกลงซื้อขายทำสำเร็จ ฟังก์ชันของเซอร์ฟเวอร์จะส่งผลการซื้อขายมาในรูปของโครงสร้างข้อมูล RETURN _TYPE (มีกล่าวรายละเอียดในส่วนเรื่องโครงสร้างข้อมูล) กลับมาให้ฟังก์ชัน Save_result ในฝั่งไคล์เอ็นต์ เพื่อบันทักข้อมูล

ฟังก์ชัน View Result ทำงานเมื่อผู้ใช้ต้องการซื้อขาย ของรายการเสนอซื้อขายที่มี Request ID ตามที่ผู้ใช้ป้อนฟังก์ชันเริ่มรับ Request ID จากผู้ใช้ แล้วนำหมายเลข Request ID มาเปรียบเทียบกับค่าตัวแปรสำหรับนับจำนวนรายการซื้อขายในขณะปัจจุบัน

- ถ้าค่า Request ID มากกว่าจำนวนรายการ จะกลับไปรอรับค่า Request ID ใหม่ แต่ถ้าผู้ใช้กดคีย์เอ็นเทอร์โดยไม่ใส่ค่าใด ๆ ฟังก์ชันจะกลับออกไปรับคำสั่งจากผู้ใช้ทันที

- ถ้าค่า Request ID น้อยกว่าหรือเท่ากับจำนวนรายการ จะเปิดไฟล์ ORDER.DAT และ RESULT.DAT เพื่ออ่านข้อมูลของ Request ID นั้น

แล้วนำมาแสดงผลในหน้าต่าง ข้อมูลที่แสดงนั้นประกอบด้วย

หมายเลข Request ID, ชื่อผู้เป็นเจ้าของรายการ, ชื่อหุ้นในรายการ, จำนวนหุ้นที่เสนอซื้อหรือขาย, จำนวนหุ้นที่สามารถตกลงซื้อหรือขายได้, จำนวนหุ้นที่คงเหลือจากการซื้อหรือขาย, ราคาต่อหุ้น, มูลค่าของการตกลงซื้อขายหุ้นในรายการนั้น (บาท), เวลาที่สร้างรายการ และ เวลาการตกลงซื้อขายครั้งล่าสุด

บทที่ 5

การทดลองและผลการทดลอง

5.1 สรุปผลการทดสอบใช้งานแอพพลิเคชัน

- กรณีที่ 1 แอพพลิเคชันนี้สามารถรองรับผู้ใช้ได้เป็นจำนวนมาก เนื่องจากการทำงานระหว่างไคล์เอ็นต์และเซอร์ฟเวอร์เป็นเพียงการเรียกฟังก์ชันและส่งอาร์กิวเมนต์ไป แล้วเซอร์ฟเวอร์ส่งผลลัพธ์กลับมาเท่านั้น แต่สำหรับในแอพพลิเคชันนี้กำหนดจำนวนผู้ใช้สูงสุด (MAX_USER) ไว้เพียง 50 ราย และการทดลองใช้งานจริงโดยทีมผู้พัฒนามีผู้ใช้เพียง 5 รายเท่านั้น เนื่องจากมีโฮสต์อยู่เพียง 5 โฮสต์ที่ติดต่อถึงกันได้ ได้แก่ lcad00 lcad01 diamond opal และ jade และต้องผู้ใช้แอพพลิเคชันแต่ละรายต้องมี account สำหรับโฮสต์ที่ต่างกัน การทดสอบนี้ทำงานโดยให้ opal ทำหน้าที่เป็นทั้งตลาดหลักทรัพย์และโบรกเกอร์ ส่วนโฮสต์ที่เหลือเป็นโบรกเกอร์ทั้งหมด การทำงานการซื้อขายหลักทรัพย์ทำงานได้ดี

- กรณีที่ 2 ถ้าไม่กำหนดให้รันฟังก์ชันของเซอร์ฟเวอร์ไว้ที่โฮสต์ของเซอร์ฟเวอร์ (ตลาดหลักทรัพย์) เมื่อรันฟังก์ชันของไคล์เอ็นต์ (บริษัทหลักทรัพย์-โบรกเกอร์) ที่เรียกใช้ฟังก์ชันของเซอร์ฟเวอร์ผ่าน RPC เคอร์เนลจะสร้างสัญญาณ (signal) illegal instruction และสร้างไฟล์ core

- กรณีที่ 3 ถ้ากำหนดให้รันฟังก์ชันของเซอร์ฟเวอร์ไว้ที่โฮสต์ของเซอร์ฟเวอร์มากกว่า 1 ครั้ง เมื่อรันฟังก์ชันของโบรกเกอร์ที่เรียกใช้ฟังก์ชันของเซอร์ฟเวอร์ผ่าน RPC ฟังก์ชันของโบรกเกอร์จะไม่สามารถทำงานต่อไปได้ ระบบจะค้างอยู่ที่ฟังก์ชันนั้น

5.2 การทดสอบหาข้อจำกัดของแอพพลิเคชันกับเครื่องที่ใช้งาน

- แอพพลิเคชันนี้ถูกพัฒนาบนเครื่อง HP ไม่สามารถคอมไพล์และทำงานบน Linux และ Sun ได้

- ฟังก์ชันของเซอร์ฟเวอร์ที่พัฒนาในกรณีศึกษานี้ใช้ Interprocess Procedure Call (RPC) ดังนั้นจึงไม่สามารถรันฟังก์ชันของเซอร์ฟเวอร์บนโฮสต์ที่ไม่มีความสามารถของ IPC System V ได้ แต่ยังสามารถรันฟังก์ชันของไคล์เอ็นต์ได้

- ความสามารถในการเรียกฟังก์ชันผ่าน RPC ขึ้นอยู่กับสมรรถนะของโฮสต์ และระยะทางระหว่างโฮสต์ไคล์เอ็นต์และโฮสต์เซอร์ฟเวอร์ กล่าวคือถ้าสมรรถนะของโฮสต์ต่ำและมีระยะทางไกล ความเร็วในการเรียกฟังก์ชันผ่าน RPC จะน้อยลงคือใช้เวลามากขึ้น

5.3 การทดลองที่ 1 การทดสอบการติดต่อระหว่างโพรเซสโดย RPC

5.3.1 วิธีการทดสอบ

เป็นการหาความเร็วในการเรียกฟังก์ชันผ่าน RPC โดยแบ่งการทำงานเป็น 3 กรณี คือ

1. การเรียกฟังก์ชันผ่าน RPC โดยฟังก์ชันของไคล์เอ็นต์และเซอร์ฟเวอร์อยู่บนโฮสต์ตัวเดียวกัน เช่น lcad01 กับ lcad01 หรือ opal กับ opal

2. การเรียกฟังก์ชันผ่าน RPC โดยฟังก์ชันของไคล์เอ็นต์และเซอร์ฟเวอร์อยู่บนต่างโฮสต์ แต่อยู่ในกลุ่มเดียวกัน เช่น opal กับ diamond

3. การเรียกฟังก์ชันผ่าน RPC โดยฟังก์ชันของไคล์เอ็นต์ของเซอร์ฟเวอร์อยู่บนต่างโฮสต์ ต่างกลุ่มกัน เช่น lcad01 กับ opal

โฮสต์ที่ใช้ดังกล่าวมานี้แบ่งโดยการเชื่อมต่อได้เป็น 2 กลุ่ม คือ กลุ่มที่ 1 ได้แก่ diamond, opal และ jade ส่วนกลุ่มที่ 2 ได้แก่ lcad00 และ lcad01 การทดสอบทำโดยเขียนฟังก์ชันไคล์เอ็นต์ที่เรียกไปยังฟังก์ชันเซอร์ฟเวอร์และใช้คำสั่ง time ของระบบเก็บเวลาการทำงานทั้งหมดในการเรียกไป และรีเทิร์นกลับ คำสั่งที่ใช้จึงเป็นดังนี้

```
$ callee &          ( โปรแกรมเซอร์ฟเวอร์ )
$ time caller       ( จับเวลาการทำงานของโปรแกรมไคล์เอ็นต์ )
```

เราจะได้รับเวลาที่ใช้ มีหน่วยเป็นวินาที ทดสอบเก็บเวลาที่ใช้ในการติดต่อระหว่างโฮสต์แต่ละคู่จำนวน 10 ครั้งแล้วหาค่าเฉลี่ยของเวลา โปรแกรมที่ใช้ในการทดสอบ ประกอบด้วย

ไฟล์ logon.x กำหนดโปรโตคอลการติดต่อที่ใช้ร่วมกัน

```
/* RPC protocol for send request to market */

typedef struct log_type log_req;
struct log_type
      { string host<10>;
        string name<10>;
        string password<10>;
      };

program REQ_PROG
```

```
    {
      version REQ_VERS
       {
       void REQ_LOGON(log_req) = 1;
       }     = 1;
    } = 0x20000001;        /* program number */
```

ไฟล์ caller.c เป็นโปรแกรมไคล์เอ็นต์

```
#include <rpc/rpc.h>
#include <stdio.h>
#include <sys/time.h>

#include "logon.h"

main()
{ log_req temp;
  CLIENT *cl;

   cl = clnt_create("diamond",REQ_PROG,REQ_VERS,"tcp");
   req_logon_1(&temp,cl);
}
```

ไฟล์ callee.c เป็นโปรแกรมเซอร์ฟเวอร์

```
#include <rpc/rpc.h>
#include <stdio.h>
#include <sys/time.h>
 #include "logon.h"

void *req_logon_1(r)
log_req *r;
{
}
```

โฮสต์ที่ใช้ทำการทดลองครั้งนี้ ได้แก่

diamond	HP9000	series 800	รุ่น 827
opal	HP9000	series 700	รุ่น 710
lcad01	HP9000	series 700	รุ่น 720

5.3.2 ผลการทดสอบ

ผลการทดสอบสำหรับแต่ละกรณี เรียงตามลำดับได้ดังนี้

1. การเรียกฟังก์ชันผ่าน RPC โดยฟังก์ชันของไคล์เอ็นต์และเซอร์ฟเวอร์อยู่บนโฮสต์ตัวเดียวกัน

1.1 diamond เรียกไปยัง diamond

ครั้งที่	เวลา(วินาที)
1	0.15
2	0.07
3	0.07
4	0.07
5	0.08
6	0.07
7	0.07
8	0.07
9	0.07
10	0.08

เวลาเฉลี่ย คือ 0.08 วินาที

1.2 lcad01 ไปยัง lcad01

ครั้งที่	เวลา (วินาที)
1	0.10
2	0.05
3	0.04

4	0.05
5	0.06
6	0.06
7	0.06
8	0.06
9	0.05
10	0.07

เวลาเฉลี่ย คือ 0.06 วินาที

2. การเรียกฟังก์ชันผ่าน RPC โดยฟังก์ชันของไคล์เอ็นต์และเซอร์ฟเวอร์อยู่บนต่างโฮสต์ แต่อยู่ในวงเดียวกัน

2.1 opal เรียกไปยัง diamond

ครั้งที่	เวลา (วินาที)
1	0.12
2	0.08
3	0.18
4	0.09
5	0.08
6	0.18
7	0.09
8	0.16
9	0.08
10	0.08

เวลาเฉลี่ย คือ 0.082 วินาที

3. การเรียกฟังก์ชันผ่าน RPC โดยฟังก์ชันของไคล์เอ็นต์ของเซอร์ฟเวอร์อยู่บนต่างโฮสต์ ต่างวงกัน

3.1 lcad01 เรียกไปยัง diamond

ครั้งที่	เวลา (วินาที)
1	0.12
2	0.08
3	0.18

4	0.09
5	0.08
6	0.18
7	0.09
8	0.16
9	0.08
10	0.08

เวลาเฉลี่ย คือ 0.114 วินาที

แสดงค่าเฉลี่ยทั้ง 3 กรณี ได้ดังนี้

กรณี	เวลา (วินาที)
1. การเรียกในโฮสต์เดียวกัน	
diamond - diamond	0.08
lcad01 - lcad01	0.06
2. การเรียกไปต่างโฮสต์ในกลุ่มเดียวกัน	
opal - diamond	0.082
3. การเรียกไปต่างโฮสต์ต่างกลุ่มกัน	
lcad01 - diamond	0.114

5.3.3 สรุปและวิจารณ์การทดสอบ

1. การเรียกฟังก์ชันผ่าน RPC ไม่ว่ากรณีใด ๆ ต้องมีการส่งอาร์กิวเมนต์ผ่านไคล์เอ็นต์สตับ (client stub) และ เซอร์ฟเวอร์สตับ (server stub) ไปยังโปรแกรมเซอร์ฟเวอร์เสมอ ดังนั้นเวลาในกรณีที่ 1, 2 และกรณีที่ 3 จึงไม่ควรแตกต่างกันมาก เวลาที่แตกต่างจึงมาจากระยะทางระหว่างโฮสต์ ,จำนวนผู้ใช้ในโฮสต์ และสมรรถนะของเครื่องเป็นสำคัญ

2. โฮสต์ opal อาจมีสมรรถนะด้อยกว่าโฮสต์ diamond เนื่องจากว่า เวลาที่บันทึกได้ของการติดต่อระหว่าง opal กับ diamond นานกว่าการติดต่อระหว่าง diamond กับ diamond ซึ่งสอดคล้องกับสมรรถนะของโฮสต์ที่ได้กล่าวไว้ข้างต้น

3. การติดต่อระหว่าง lcad01 กับ diamond ใช้เวลามากที่สุด เนื่องจาก โฮสต์ทั้งสองมีระยะทางห่างกันมากที่สุดในสามกรณี และมีผู้ใช้งานใน lcad01 อยู่เป็นจำนวนมากในขณะที่ทำการทดสอบ

4. เวลาที่ได้จากการบันทึกผลทุกค่ามีค่าใกล้เคียงกัน และ เวลาเฉลี่ยที่ได้ทั้งของ 3 กรณี คือ 0.084 วินาที

5.4 การทดลองที่ 2 การเปรียบเทียบการติดต่อแบบ RPC และ IPC

5.4.1 วิธีการทดสอบ

เป็นการจับเวลาการส่งข้อมูลที่มีขนาดเท่ากันโดยวิธีแบบ RPC และวิธี IPC ชนิด เมสเสจคิว (Message Queue) บนโฮสต์ตัวเดียวกัน คือ โฮสต์ jade การทดสอบนี้ทำโดยใช้กรณีที่ข้อมูลมีขนาดต่างกัน 3 ค่า ได้แก่

1. การส่งข้อมูลขนาด 1 ไบต์ โดยวิธี RPC และโดยใช้เมสเสจคิว
2. การส่งข้อมูลขนาด 1 กิโลไบต์ (1024 ไบต์) โดยวิธี RPC และโดยใช้เมสเสจคิว
3. การส่งข้อมูลขนาด 4 กิโลไบต์ (4096 ไบต์) โดยวิธี RPC และโดยใช้เมสเสจคิว

การทดลองการส่งข้อมูลทั้ง 3 กรณี เก็บเวลาโดยใช้ฟังก์ชัน time ของระบบเช่นเดียวกับการทดลองที่ 1 และเก็บเวลาการส่งข้อมูลกรณีละ 10 ครั้งแล้วนำมาคำนวณค่าเฉลี่ย

โปรแกรมไคล์เอ็นต์ และโปรแกรมเซอร์ฟเวอร์ที่ใช้ในการทดสอบการส่งข้อมูลแบบ RPC นี้มีการทำงานเช่นเดียวกับโปรแกรมที่ใช้ในการทดลองที่ 1 แต่กำหนดตัวแปรข้อมูลที่ส่งเป็น

```
struct log_type
    { string temp<1>;
    };
```

โดยขนาดของสตริง temp เปลี่ยนไปตามกรณีทดสอบ คือ 1 ไบต์, 1024 ไบต์ และ 4096 ไบต์

โปรแกรมที่ใช้สำหรับทดสอบการส่งข้อมูลโดยเมสเสจคิว ประกอบด้วยโปรแกรมฝ่ายส่งข้อมูล และโปรแกรมฝ่ายรับข้อมูล ดังนี้

โปรแกรมฝ่ายส่งข้อมูล

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define  MSGKEY        75

sturct    msgform {
```

```
        long    mtype;
        char    mtext[1];
} msg;
int     msgid;
msin()
{       msgid = gsgget(MSGKEY, 0777);
        msgsnd(msgid, &msg, sizeof(int), 0);
}
```

โปรแกรมฝ่ายรับข้อมูล

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define  MSGKEY      75

sturct   msgform {
        long    mtype;
        char    mtext[1];
} msg;
int     msgid;

msin()
{       msgid = gsgget(MSGKEY, 0777 | IPC_CREAT);
        msgrcv(msgid, &msg, sizeof(int), 0);
}
```

ขนาดของ mtext ในโปรแกรม คือ ขนาดของข้อมูลที่ส่งผ่านเมสเสจคิว เปลี่ยนแปลงได้ตามกรณีทดสอบ คือ 1 ไบต์, 1024 ไบต์ และ 4096 ไบต์

5.4.2 ผลการทดสอบ

กรณี	เวลาโดยเฉลี่ย (วินาที)	
	โดยวิธี RPC	โดยวิธี IPC (Message Queue)
1. การส่งข้อมูลขนาด 1 ไบต์	0.072	0.028
2. การส่งข้อมูลขนาด 1024 ไบต์	0.075	0.031
3. การส่งข้อมูลขนาด 4096 ไบต์	0.078	0.028

5.4.3 สรุปผลและวิจารณ์การทดลอง

1. การส่งข้อมูลไปยังฟังก์ชันโดยผ่าน RPC ไม่ว่ากรณีใด ๆ ต้องมีการส่งอาร์กิวเมนต์ผ่าน**ไคล์เอ็นต์สตับ** (client stub) และ เซอร์ฟเวอร์สตับ (server stub) ไปยังโปรแกรมเซอร์ฟเวอร์เสมอ ดังนั้นจึงใช้เวลามากกว่าการส่งข้อมูลผ่าน IPC ด้วยเมสเสจคิวทุกกรณีที่ทดสอบ

2. เวลาที่ใช้สำหรับการส่งข้อมูลไปยังฟังก์ชันโดยวิธี RPC เพิ่มขึ้นเมื่อขนาดข้อมูลที่ส่งเพิ่มขึ้น

3. โฮสต์ jade ขณะใช้ทำการทดสอบนี้มีผู้ใช้เพียงรายเดียวคือผู้ทำการทดสอบ แต่จำนวนครั้งการเก็บเวลาอาจยังไม่เพียงพอต่อความถูกต้องของข้อมูล

บทที่ 6
สรุปและบทวิจารณ์

สรุป

การพัฒนาระบบจำลองการซื้อขายหลักทรัพย์ของตลาดหลักทรัพย์นี้ มีการทำงานแบบเรียลไทม์ และมีผู้ใช้ซึ่งทำหน้าที่เป็นโบรกเกอร์ได้หลายคน ผู้ใช้สามารถส่งรายการเสนอซื้อขายหลักทรัพย์ไปยังตลาด หรือสามารถส่งรายการร้องขอข้อมูลจากตลาด เช่น ราคาหลักทรัพย์ และมูลค่าการซื้อขายหลักทรัพย์ได้ รายการเสนอซื้อขายและคำร้องขอต่าง ๆ ที่ส่งให้แก่ตลาดได้เหล่านี้จะต้องมีความถูกต้องครบถ้วนระดับหนึ่ง ตามข้อกำหนดของแอพพลิเคชันที่เครื่องของผู้ใช้ หรือไคล์เอ็นต์ ภายหลังจากผ่านการตรวจสอบความถูกต้องนี้แล้ว รายการและคำร้องขอต่าง ๆ จะถูกส่งผ่านด้วยลักษณะของรีโมตโพรซีเจอร์คอล (RPC) เพื่อให้ตลาดหรือเซอร์ฟเวอร์ได้รับรายการเสนอซื้อขายและคำร้องขอต่าง ๆ นี้ไปประมวลผล

เซอร์ฟเวอร์มีความสามารถทำงานประมวลผลรายการจากผู้ใช้ได้สอดคล้องกับงานที่เข้ามาจากผู้ใช้ต่าง ๆ และสามารถควบคุมความถูกต้องในการประมวลผลได้แม้มีผู้ขอใช้บริการเข้ามามาก ทั้งนี้เพราะอาศัยการควบคุมการลำดับงาน ด้วยเมสเสจคิว (Message Queue) ซึ่งเป็นวิธีการหนึ่งของอินเตอร์โพรเซสคอมมิวนิเค่ชัน (IPC)

บทวิจารณ์

การพัฒนาแอพพลิเคชันบนระบบยูนิกซ์เพื่อใช้ในลักษณะการประมวลผลแบบกระจาย จำเป็นต้องอาศัยแนวความคิดและการออกแบบที่แตกต่างออกไปจากการพัฒนาแอพพลิเคชันทั่ว ๆ ไป เช่น แอพพลิเคชันบนดอส ดังนั้นจึงมีโอกาสพบปัญหาได้ค่อนข้างมาก

ระบบจำลองการซื้อขายหลักทรัพย์ที่พัฒนานี้ มีความสามารถในการทำงานได้เพียงส่วนหนึ่งของระบบซื้อขายหลักทรัพย์จริง เป็นต้นว่า ในแง่ของฮาร์ดแวร์ ระบบซื้อขายหลักทรัพย์ที่ใช้ในตลาดหลักทรัพย์แห่งประเทศไทย สามารถเชื่อมต่อได้กว้างขวางมาก ในขณะที่ระบบจำลองในโครงงานนี้เป็นระบบที่อยู่ในเครือข่ายของยูนิกซ์ และมีฐานะเป็นโฮสต์เท่านั้น และในแง่ฟังก์ชันของซอฟท์แวร์ ระบบซื้อขายหลักทรัพย์ในตลาดจริงมีความสามารถเพิ่มเติม เช่น การยกเลิกรายการเสนอซื้อขาย เป็นต้น ในขณะที่ในระบบจำลองที่พัฒนายังทำงานไม่ครอบคลุมฟังก์ชันนี้

สาเหตุที่เราไม่พัฒนาการทำงานให้มีฟังก์ชันสมบูรณ์เหมือนกับระบบตลาดหลักทรัพย์จริง เนื่องจากระบบจำลองนี้ ไม่ได้มีวัตถุประสงค์เพื่อเลียนแบบการซื้อขายหลักทรัพย์ให้เหมือนจริงมากที่สุด แต่ต้องการให้เป็นกรณีศึกษาการประมวลผลผ่านเครือข่าย การประมวลผลแบบกระจาย และการควบคุมการทำงานระหว่างโพรเซส

ข้อเสนอแนะและแนวทางการพัฒนาต่อ

ดังที่ได้กล่าวมาแล้วว่า ระบบจำลองนี้ยังมีความสามารถไม่สมบูรณ์เทียบเท่ากับระบบจริง ระบบจำลองนี้จึงยังคงพัฒนาเพิ่มความสามารถต่อไปได้อีกถ้าต้องการ ซึ่งอาจได้เรียนรู้ความสามารถของ RPC เพิ่มมากขึ้นและอาจรู้ข้อจำกัดได้มากขึ้นเช่นกัน

แอพพลิเคชันนี้ทำงานอยู่บนเท็กซ์โหมดของยูนิกซ์ จึงยังสามารถพัฒนาเพื่อให้ทำงานบนกราฟฟิกโหมด คือ X Windows ได้ เพื่อเพิ่มความน่าใช้และเพิ่มความสะดวกที่จะพึงได้จากระบบหน้าต่าง

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ ได้เกิดขึ้นและสำเร็จไปด้วยดี คณะผู้จัดทำต้องขอขอบพระคุณทุกท่านดังนี้

- อาจารย์ กฤตวัน เครือตราชู และ ดร. บุญธีร์ เครือตราชู ซึ่งเป็นอาจารย์ที่ปรึกษาที่ได้ให้คำแนะนำ ความรู้ต่างๆ ที่เป็นประโยชน์ในการศึกษา และวิจัยโครงงานนี้ ขอขอบคุณสำหรับความเอาใจใส่ดูแลโครงงานนี้จึงสำเร็จไปด้วยดี

- อาจารย์ อภิเนตร อูนากูล และอาจารย์ วัชระ ฉัตรวิริยะ ที่ได้ให้คำปรึกษา คำแนะนำ

- คณะผู้จัดทำ ขอกราบขอบพระคุณคุณพ่อ คุณแม่ ที่ให้ทั้งเวลา สนับสนุนในการศึกษา และกำลังใจ ตลอดจนความดูแลเอาใจใส่ที่มีตลอดมา

- ขอขอบคุณ คุณพรเทพ นฤหล้า ที่ช่วยให้คำปรึกษา และเสนอแนะแนวทางในการทำปริญญานิพนธ์นี้

- ขอขอบคุณเพื่อนๆ ทุกคนที่มีส่วนร่วม และคอยให้กำลังใจ ให้ความช่วยเหลือ และคำแนะนำ ในคราที่คณะผู้จัดทำประสบปัญหาต่างๆ

- บุคคลทุกท่านที่ทำให้ Internet เกิดขึ้นและเป็นแหล่งข้อมูลทางการศึกษาที่ยิ่งใหญ่ที่สุด

คณะผู้จัดทำ

เอกสารอ้างอิง

1. สิทธิพงศ์ จันแสงอรุณ, "ก้าวสู่เทคโนโลยีไคล์เอ็นต์เซอร์ฟเวอร์", ฉบับที่ 112, 2537, หน้า 227-233
2. เสรี สีลาลัย, สำนักพิมพ์ เบี้ยฟ้า , 214 หน้า , 2535
3. อดินุช กาญจนพิบูล, "การเงินและการธนาคาร", ภาควิชาภาษาและสังคม คณะครุศาสตร์อุตสาหกรรม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,163-168 หน้า, 2532
4. Englewood Cliffs, "UNIX System V: Programmer's guide", Prentice - Hall, Inc., 832 p ,1987
5. Goodheart Berry, "UNIX Curses explained ", Prentice - Hall, Inc., 287 p ,1991
6. John Bloomer, "Power Programming with RPC", O'Reilly & Associates, Inc., 486 p, 1992
7. Maurice J.Bach, "The Design of the Unix Operating System",Prentice - Hall, Inc., 772 p., 1990
8. W. Richard Stevens, "UNIX Network Programming", Prentice - Hall, Inc., 772 p., 1990