ระบบเฝ้าระวัง เตือนภัยและอพยพโพรเซส

SYSTEM PROCESS MONITORING ALERT AND MIGRATION

พิชญ์สินี เปลี่ยนสมัย
วรัญญู บุตรสิงขรณ์
สุรีย์กาญจน์ บุญเดช

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2553

ปริญญานิพนธ์ปีการศึกษา 2553

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส

SYSTEM PROCESS MONITORING ALERT AND MIGRATION

ผู้จัดทำ

1. นางสาวพิชญ์สินี เปลี่ยนสมัย รหัสนักศึกษา 50011373
2. นายวรัญญู บุตรสิงขรณ์ รหัสนักศึกษา 50011387
2. นางสาวสุรีย์กาญจน์ บุญเดช รหัสนักศึกษา 50011774

อาจารย์ที่ปรึกษา

(ดร.ปกรณ์ วัฒนจตุรพร)

ระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส

นางสาวพิชญ์สินี	เปลี่ยนสมัย	50011373
นายวรัญญู	บุตรสิงขรณ์	50011387
นางสาวสุรีย์กาญจน์	บุญเดช	50011774
ดร.ปกรณ์	วัฒนจตุรพร	อาจารย์ที่ปรึกษา

ปีการศึกษา 2553

บทคัดย่อ

ระบบเฝ้าระวัง และเตือนภัยในปัจจุบัน มีแนวคิดเพียงการแจ้งเตือนให้กับผู้ดูแลระบบเท่านั้น แต่ไม่ได้มีการแก้ไขปัญหาที่เกิดขึ้น ทำให้ผู้ดูแลจำเป็นที่จะต้องมีการเฝ้าระวังอยู่ตลอดเวลา เมื่อเกิดปัญหา ผู้ดูแลก็จะต้องหาสาเหตุ หรือที่มาของปัญหาด้วยตัวเอง ซึ่งส่งผลให้การดำเนินการแก้ไขเป็นไปอย่างล่าช้า นอกจากนี้ยังทำให้ประสิทธิภาพโดยรวมของระบบแย่ลง เพราะไม่สามารถให้บริการได้อย่างต่อเนื่อง เนื่องผู้ดูแลระบบจำเป็นต้องแบกภาระทั้งหมดในการเฝ้าระวังระบบเหล่านี้ การสร้างระบบที่สามารถแก้ไขปัญหาเบื้องต้น หรือระบบที่สามารถกระจายภาระงานให้กับเครื่องเซิร์ฟเวอร์อื่นๆให้สมดุลกัน เพื่อให้บริการอย่างต่อเนื่องได้โดยที่ไม่ทำให้ประสิทธิภาพโดยรวมของระบบลดลง จะทำให้ภาระงานของผู้ดูแลระบบในส่วนของการเฝ้าระวังน้อยลง ซึ่งส่งผลให้ผู้ดูแลสามารถให้ความสำคัญกับงานด้านอื่นเพิ่มขึ้น และส่งผลให้งานโดยรวมที่ผู้ดูแลมีส่วนรับผิดชอบอยู่มีประสิทธิผลที่ดียิ่งขึ้น โดยระบบนี้จะมีแนวคิดในการแบ่งเอเจนท์โพรเซส และเมเนเจอร์โพรเซส เพื่อทำหน้าที่ที่ต่างกันไป กล่าวคือ เอเจนท์โพรเซสจะฝังอยู่กับเซิร์ฟเวอร์ที่เราต้องการจะเฝ้าระวัง ส่วนเมเนเจอร์โพรเซสจะฝังอยู่กับเซิร์ฟเวอร์ที่เราใช้สำหรับเก็บข้อมูลทุกอย่างที่รับมาจากเอเจนท์ รวมถึงการแสดงค่าต่างๆเพื่อให้สามารถเฝ้าระวังได้ง่ายขึ้น เมื่อเกิดเหตุการณ์ที่ผิดปกติ เอเจนท์จะสามารถประเมินความเสียหายที่เกิดขึ้น และแก้ไขปัญหาเบื้องต้นเหล่านี้โดยอัตโนมัติ เพื่อให้ระบบสามารถดำเนินการต่อไปได้อย่างมีประสิทธิภาพ

System Process Monitoring Alert and Migration

Miss.	Pitchsinee	Piansamai	50011373
Mr.	Waranyoo	Butsingkorn	50011387
Miss.	Sureekarn	Boondej	50011774
Dr.	Pakorn	Watanachaturaporn	Advisor

Academic Year 2010

ABSTRACT

Most monitoring systems function on monitoring system status and alert an administrator to fix problems. The administrator has to keep monitoring the systems at all time and find the source of problem when it occurs. Quite often it takes a long time to find a cause of the problem and to fix it. These decrease an efficiency of the whole system. This project aims to assist the administrator by having an agent and a manager process resided in the system. Each machine needs one agent to monitoring the status, and the system needs one manager process to monitor the whole system. The agent process helps the administrator to fix a simple problem. Moreover, it monitors and detects anomalous situations. If the server has been highly loaded, the agent will detect and assess the level of risk. In a situation that the system load is higher than a preset level the manager will be notified. The manager will distribute the load to a lighter load machine without the help from the administrator. Therefore, the system does not have to wait for the administrator to find a cause of problem, and the overall efficiency of the system is increased.

กิตติกรรมประกาศ

ระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสไม่อาจสำเร็จได้หากไม่ได้รับความร่วมมือจากหลายๆฝ่ายซึ่งช่วยกันดูแล สนับสนุน และผลักดันนักศึกษาให้ดำเนินงานไปจนสำเร็จลุล่วงด้วยดี

ทั้งนี้ขอขอบคุณสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง สาขาวิชาวิศวกรรมคอมพิวเตอร์ รวมทั้งสถานศึกษาในอดีตของนักศึกษา ที่เอื้อเฟื้อโอกาสทางการศึกษา และเป็นแหล่งความรู้ซึ่งทำให้นักศึกษาได้เกิดการพัฒนาตนเองเรื่อยมา

ขอขอบคุณสำนักบริการคอมพิวเตอร์ สำหรับสถานที่ในการเพิ่มโอกาสการศึกษา อุปกรณ์สำหรับการทำการศึกษาและวิจัยในหัวข้อต่างๆ รวมทั้งสถานที่สำหรับการทำงาน ทั้งนี้ต้องขอขอบพระคุณเจ้าหน้าที่ พนักงาน และผู้อำนวยการสำนักบริการคอมพิวเตอร์ทุกท่าน ที่เอื้อเฟื้อสิ่งเหล่านี้ให้กับข้าพเจ้า ทำให้ข้าพเจ้ามีความรู้ความสามารถเพิ่มพูนมากขึ้นเป็นระยะเวลาถึง 4 ปี

ขอขอบพระคุณอาจารย์ปกรณ์ วัฒนจตุรพร อาจารย์ที่ปรึกษาร่วมซึ่งค่อยแนะนำ ชี้แนะข้อบกพร่อง และแนะนำแนวทางแก้ไขปัญหาที่เกิดขึ้น รวมทั้งให้คำปรึกษาในการทำรายงานระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสครั้งนี้สำเร็จลุล่วงไปด้วยดี

ขอขอบคุณนายปรมินทร์ อินโสม และนายโชคชัย พุฒเตาล สำหรับคำแนะนำ และแนวทางการทำระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสครั้งนี้ให้สำเร็จลุล่วงไปด้วยดี

และสุดท้าย ขอขอบพระคุณบุคคลที่มีความสำคัญมากที่สุดในชีวิต คือ บิดา มารดา และผู้มีพระคุณอื่นๆที่ทำให้นักศึกษาสามารถก้าวมาได้จนถึงปัจจุบันนี้ ไม่ว่าจะด้วยการเลี้ยงดู ให้โอกาสทางการศึกษา ชี้แนะแนวทาง หรือให้กำลังใจในการทำงาน นักศึกษาระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณทุกๆท่านมา ณ ที่นี้

นางสาวพิชญ์สินี เปลี่ยนสมัย
นายวรัญญู บุตรสิงขรณ์
นางสาวสุรีย์กาญจน์ บุญเดช

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

รูป หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาของปัญหา

ระบบเฝ้าระวัง และเตือนภัยส่วนใหญ่ในปัจจุบัน มักมีแนวคิดในการแจ้งเตือนเมื่อเกิดปัญหาเท่านั้น ทำให้ผู้ดูแลระบบจำเป็นต้องให้การดูแล และเฝ้าระวังอยู่ตลอดเวลา นอกจากนี้เมื่อมีปัญหาเกิดขึ้น การแก้ไขปัญหาบางจุดอาจกระทำได้อย่างล่าช้า เนื่องจากผู้ดูแลต้องทำการค้นหาต้นตอหรือสาเหตุของการเกิดปัญหาเหล่านั้นด้วยตัวเอง ส่งผลให้ประสิทธิภาพการทำงานโดยรวมของระบบแย่ลง เพราะเซิร์ฟเวอร์ หรือระบบที่มีปัญหาจะไม่สามารถให้บริการได้ นอกเสียจากว่าระบบนั้นจะมีการวางระบบซ้ำซ้อน หรือระบบสำรองเพื่อให้บริการในสภาวะที่ระบบหลักมีปัญหา ซึ่งค่าใช้จ่ายในการสร้างระบบเหล่านี้ค่อนข้างสูง ส่งผลให้ไม่ค่อยเป็นที่นิยมสำหรับกลุ่มลูกค้าขนาดกลาง และล่างเท่าไรนัก

การพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสจึงเกิดขึ้น เพื่อช่วยลดภาระงานให้กับผู้ดูแลระบบ โดยระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสจะไม่ทำหน้าที่เพียงการแจ้งเตือนเมื่อเกิดปัญหาเท่านั้น แต่ยังสามารถที่จะแก้ไขปัญหาเบื้องต้นที่เกิดขึ้นภายในระบบได้อย่างอัตโนมัติ โดยที่ผู้ดูแลระบบอาจจะไม่จำเป็นต้องให้การดูแล และเฝ้าระวังระบบตลอดเวลาอีกต่อไป นอกจากนี้หากเกิดปัญหาจนทำให้ระบบไม่สามารถให้บริการได้ ระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสยังสามารถกระจายภาระงานของเซิร์ฟเวอร์ให้สมดุล ซึ่งจะช่วยทำให้การให้บริการของเซิร์ฟเวอร์ หรือระบบนั้นดำเนินการได้อย่างต่อเนื่องโดยไม่ทำให้ประสิทธิภาพการทำงานโดยรวมของระบบแย่ลง

1.2 วัตถุประสงค์ของโครงงาน

1) เพื่อศึกษาการทำงานของระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสสำหรับเซิร์ฟเวอร์ที่มีระบบปฏิบัติการลีนุกซ์ภายในเครือข่าย
2) เพื่อศึกษาการออกแบบระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสสำหรับเซิร์ฟเวอร์ที่มีระบบปฏิบัติการลีนุกซ์ภายในเครือข่าย
3) เพื่อเป็นแนวทางในการพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสสำหรับเซิร์ฟเวอร์ที่มีระบบปฏิบัติการลีนุกซ์ภายในเครือข่าย
4) เพื่อพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสสำหรับเซิร์ฟเวอร์ที่มีระบบปฏิบัติการลีนุกซ์ภายในเครือข่าย

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1) มีความรู้ความเข้าใจการทำงานของระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส
2) สามารถออกแบบระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสได้
3) สามารถพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสให้ใช้งานจริง

1.4 ขอบเขตของโครงงาน

1) ศึกษาการทำงานของระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสสำหรับเซิร์ฟเวอร์ที่มีระบบปฏิบัติการลีนุกซ์ภายในเครือข่ายเดียวกัน
2) ศึกษาการออกแบบ และพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสสำหรับเซิร์ฟเวอร์ที่มีระบบปฏิบัติการลีนุกซ์ภายในเครือข่ายเดียวกัน
3) ออกแบบ และพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสสำหรับเซิร์ฟเวอร์ที่มีระบบปฏิบัติการลีนุกซ์ภายในเครือข่ายเดียวกัน

1.5 ขั้นตอนการดำเนินงาน

1) ศึกษาโครงสร้างพื้นฐานของระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส
2) ศึกษาโครงสร้างพื้นฐาน การใช้คำสั่ง และการจัดการระบบปฏิบัติการลินุกซ์
3) ศึกษาวิธีการออกแบบระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส
4) ศึกษาเทคโนโลยี และภาษาที่ใช้ในการพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส
5) ออกแบบ และพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส

1.6 ส่วนประกอบของโครงงาน

รายงานฉบับนี้ประกอบด้วยเนื้อหา 5 บท ซึ่งแต่ละบทกล่าวถึงเรื่องดังต่อไปนี้
บทที่ 1 กล่าวถึงความเป็นมาของปัญหา วัตถุประสงค์ของโครงงาน ประโยชน์ที่คาดว่าจะได้รับ ขอบเขตของโครงงาน ขั้นตอนการดำเนินงาน และส่วนประกอบของรายงาน
บทที่ 2 กล่าวถึงทฤษฎีเบื้องต้นในการพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส
บทที่ 3 กล่าวถึงระบบปฏิบัติการลีนุกซ์
บทที่ 4 กล่าวถึงการออกแบบระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส
บทที่ 5 กล่าวถึงภาษาเพิร์ล
บทที่ 6 กล่าวถึงเครื่องมือช่วยในการพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 ความหมายของระบบเฝ้าระวัง

ระบบเฝ้าระวัง คือเครื่องมือเฝ้าระวัง ตรวจสอบ วิเคราะห์ระบบ และวัดประสิทธิภาพของระบบ ซึ่งเป็นเครื่องมือตรวจสอบประสิทธิภาพของระบบเพื่อให้ผู้ดูแลระบบสามารถดำเนินงานต่อไปได้โดยไม่เกิดปัญหา ซึ่งเป็นเครื่องมือที่ช่วยลดภาระให้แก่ผู้ดูแลระบบ

2.2 พื้นฐานคอมพิวเตอร์เน็ตเวิร์ค

2.2.1 ทีซีพีไอพีโมเดล (TCP/IP Model)

ทีซีพีไอพีโมเดลเป็นมาตรฐานที่ทำให้คอมพิวเตอร์ภายในระบบเครือข่ายอินเทอร์เน็ต สามารถเชื่อมต่อเข้าหากัน และติดต่อสื่อสารแลกเปลี่ยนข้อมูลกันได้ เป็นมาตรฐานที่ว่าด้วยการกำหนดวิธีการติดต่อสื่อสารระหว่างคอมพิวเตอร์ โดยใช้แนวคิดของการแบ่งลำดับชั้นโปรโตคอล ลำดับชั้นของโปรโตคอลในระบบอินเทอร์เน็ต มีลำดับชั้นที่น้อยกว่าโครงสร้างลำดับชั้นของโอเอสไอ โดยในโอเอสไอมีลำดับชั้นของโปรโตคอลทั้งหมด 7 ชั้น แต่ในระบบอินเทอร์เน็ตมีทั้งหมดเพียง 4 ชั้น ดังนี้

2.2.1.1 Network Interface

สามารถเทียบได้กับชั้นที่ 1 และ 2 ในโครงสร้างแบบโอเอสไอโมเดล ลำดับชั้นที่ไม่ได้เกี่ยวข้องกับระบบอินเทอร์เน็ตโดยตรง แต่เป็นระบบพื้นฐานของการเชื่อมต่อที่ระบบอินเทอร์เน็ตใช้ส่งข้อมูลภายในเครือข่าย หน้าที่ของชั้นนี้สำหรับการส่งข้อมูล โดยจัดเตรียมข้อมูลเพื่อให้เหมาะสมก่อนที่จะส่งไปตามสายส่งไปยังที่หมายปลายทาง ซึ่งได้แก่การจัดเตรียมส่วนหัวของแพ็กเกจการควบคุมระบบฮาร์ดแวร์ และซอฟต์แวร์ที่จะใช้ในการจัดส่ง เช่น การเชื่อมต่อกับเน็ตเวิร์คการ์ด และการใช้งานดีไวท์ไดร์เวอร์ หน้าที่สำหรับการรับข้อมูล คือคอยรับข้อมูลแล้วนำข้อมูลส่วนหัวออก และจัดเตรียมข้อมูลเพื่อส่งต่อไปยังชั้นเครือข่ายต่อไป

2.2.1.2 Internet

เทียบได้กับชั้นที่ 3 คือชั้น Network ในโครงสร้างแบบโอเอสไอโมเดล เป็นชั้นที่มีหน้าที่ส่งข้อมูลจากจุดเริ่มต้นไปยังปลายทาง โดยหาเส้นทางที่ข้อมูลจะใช้เดินทางผ่านเครือข่ายหนึ่งไปยังอีกเครือข่ายหนึ่งจนกระทั่งถึงปลายทาง

โปรโตคอลที่ใช้ในชั้นนี้ Internet Protocol หรือ IP ทำหน้าที่เปรียบเสมือนซองจดหมายซึ่งระบุถึงที่อยู่ของต้นทาง และปลายทาง โดยมีบุรุษไปรษณีย์ทำหน้าที่ส่งจดหมายนั้นผ่าน

สามารถส่งข้อมูลได้เร็ว และไม่มีการสร้างการเชื่อมต่อเหมือนทีซีพีโปรโตคอล ทำให้ข้อมูลที่วิ่งในเครือข่ายมีน้อยลงด้วย

ยูดีพีมีจุดเด่นที่ความเร็ว ขนาดเล็ก และไม่มีการทำงานเกี่ยวการส่งข้อมูลซ้ำ หรือคำนวณอัตราการส่งข้อมูล ซึ่งเหมาะกับการส่งข้อมูลแบบเรียลไทม์ ซึ่งข้อมูลที่สูญหายบางส่วนหรือข้อมูลที่ล่าช้าจะถูกละความสนใจไป โดยการสื่อสารแบบไม่ต้องมีการเชื่อมต่อก่อน ข้อมูลจะถูกแบ่งเป็นชิ้นๆตามที่อยู่ปลายทาง แล้วผ่านตัวกลางไปยังปลายทาง อาจจะใช้เส้นทางคนละเส้นทางกันก็ได้ รวมทั้งข้อมูลแต่ละชิ้นอาจจะถึงก่อนหลังแตกต่างกันไปได้ด้วย ทำให้การเริ่มต้นส่งทำได้รวดเร็ว ไม่ต้องเสียเวลาสร้างการเชื่อมต่อ

2.3 การอพยพโพรเซส (Process Migration)

การอพยพโพรเซส คือการย้ายโพรเซสระหว่างเครื่องคอมพิวเตอร์ 2 เครื่อง โดยหลักการของการอพยพโพรเซสนั้น จะสามารถทำให้เกิดการทำงานใน 4 รูปแบบ คือ ทำให้เกิดการกระจายโหลดแบบพลวัต (Dynamic Load Distributed) ลดโอกาสในการทำงานผิดพลาดของเครื่องคอมพิวเตอร์ (Fault Resilience) ลดภาระงานในการดูแลเครื่อง (Improved System Administration) และสามารถที่จะเข้าถึงข้อมูลต่างๆได้อย่างอิสระ (Data Access Locality) หากขาดไปข้อใดข้อหนึ่งก็ไม่อาจเรียกว่าเป็นการอพยพโพรเซสได้ นักวิจัยได้มีความพยายามในการทำวิจัยเรื่องของการอพยพโพรเซสเป็นอย่างมาก แต่ก็พบว่าไม่ประสบความสำเร็จเท่าที่ควร จนกระทั่งระบบปฏิบัติการ และเทคโนโลยีการประมวลแบบกระจายได้เข้ามามีบทบาทในการทำงานทางวิทยาศาสตร์มากขึ้น ทำให้แนวคิดนี้กลับมาเป็นที่สนใจทั้งในรูปแบบของการวิจัยและเกิดการสร้างผลิตภัณฑ์เพื่อการวิจัยทางวิทยาศาสตร์

ถ้าจะกล่าวถึงสาเหตุหลักที่ทำให้การอพยพโพรเซสไม่เป็นที่สนใจเท่าที่ควรในช่วงแรกเริ่มนั้นอาจเนื่องมาจากความซับซ้อนในการพัฒนา และการออกแบบระบบให้สามารถรองรับการอพยพโพรเซสสำหรับระบบเก่า นอกจากนี้ยังมีตัวแปรที่สำคัญ คือ ผู้ใช้งาน(User) ผู้ใช้งานคอมพิวเตอร์ส่วนใหญ่ในขณะนั้นไม่ได้มีความต้องการที่จะใช้ระบบที่ซับซ้อนหรือมีขนาดใหญ่มาก แต่จะเน้นในเรื่องของการใช้งานเล็กๆ เช่น การทำงานเอกสาร (Word Processing) มีการทำงานง่ายๆ ซึ่งการใช้งานระบบปฏิบัติการแบบกระจายจะไม่ช่วยทำให้ผู้ใช้งานทำงานได้เร็วขึ้น นอกจากนี้ยังต้องอาศัยความเข้าใจในเรื่องของการทำงานภายในของระบบที่มีความซับซ้อนมากในการใช้งานระบบอีกด้วย

การอพยพโพรเซสนั้นสามารถเกิดขึ้นได้ทั้งในขณะที่โพรเซสกำลังจะถูกสร้าง หรือโพรเซสถูกสร้างจนกระทั่งทำงานเป็นระยะเวลาหนึ่งแล้วก็ได้ และการอพยพโพรเซสครั้งหนึ่ง เราจำเป็นจะต้องทราบว่าโพรเซสตัวไหนที่จะถูกอพยพบ้าง สถานที่อพยพของโพรเซสอยู่ที่ไหน และเมื่อไร

เราจึงต้องทำการอพยพ (What Where and When) กำหนดขึ้นมาเป็นนโยบายสำหรับการอพยพโพรเซสให้ชัดเจนก่อนที่จะทำการอพยพโพรเซสนั้นๆ ซึ่งการอพยพโพรเซส เราจำเป็นต้องอาศัยกลไกการอพยพโพรเซสเข้ามาช่วยในการจัดการการถ่ายโอนโพรเซสจากโหนดหนึ่งไปยังอีกโหนดหนึ่ง เพื่อให้สามารถอพยพโพรเซสได้อย่างมีประสิทธิภาพ

ประโยชน์ที่เกิดขึ้นจากการอพยพโพรเซสในระบบประมวลผลแบบกระจาย มีดังนี้

1) เกิดการจัดงานให้สมดุล (Load Balancing) เพราะทำให้โพรเซสสามารถถูกกระจายไปยังโหนดที่มีอัตราการว่างงานอยู่ ทำให้การทำงานของระบบโดยรวมมีประสิทธิภาพมากขึ้น
2) เกิดความคงทนต่อการเสียหาย (Fault Tolerance) โพรเซสที่มีการทำงานเหมือนกัน สามารถกระจายตัวไปตามโหนดต่างๆ เพื่อให้สามารถประมวลผลแบบขนาน ส่งผลให้อัตราการเกิดความเสียหาย หรือผิดพลาดจากการทำงานของโพรเซสลดลง
3) ระบบมีความน่าเชื่อถือ (Reliability) เนื่องจากโพรเซสสามารถถูกอพยพไปยังโหนดอื่นๆ เมื่อโหนดกำลังเกิดปัญหา หรืออยู่ในสภาวะซ่อมบำรุง
4) เกิดสภาวะการทำงานพร้อมๆกัน (Concurrency) แอปพลิเคชั่นสามารถออกแบบให้รองรับกับการอพยพโพรเซส เพื่อให้โพรเซสเหล่านั้นทำงานพร้อมกันได้อย่างมีประสิทธิภาพ
5) เกิดสภาวะความพร้อมใช้งานสูงสุด (Availability) เราสามารถสร้างแหล่งข้อมูลสำหรับการประมวลผลของโพรเซสไว้ในหลายๆ โหนด เพื่อรองรับการอพยพโพรเซสไปยังโหนดนั้นๆ ส่งผลให้อัตราการคัดลอกสถานะและข้อมูลต่างๆที่โพรเซสจำเป็นต้องใช้งานลดลง
6) ลดภาระงานในระบบเครือข่าย (Reduced network traffic) โพรเซสอาจถูกอพยพไปยังโหนดที่มีการเปิดการใช้งานระบบเครือข่ายอย่างเต็มรูปแบบ ซึ่งอาจทำให้สามารถลดปริมาณเครือข่ายที่เข้ามายังโพรเซสนั้นได้
7) เกิดการจำกัดการเข้าถึงข้อมูล (Remote resource access) โพรเซสอาจถูกอพยพไปยังโหนดที่มีการจำกัดการเข้าถึงข้อมูลทำให้เกิดความปลอดภัยมากขึ้น

ข้อเสียจากการอพยพโพรเซสที่เกิดขึ้นในระบบประมวลผลแบบกระจาย มีดังนี้

1) มีความซับซ้อน (Additional complexity) นโยบายการอพยพโพรเซสและกลไกที่ต้องใช้ในการอพยพโพรเซสนั้นจะมีความซับซ้อนมาก การออกแบบจะขึ้นอยู่กับความต้องการระบบ
2) เกิดโอเวอร์เฮด (Additional overhead) เพราะโพรเซสอื่นๆที่ไม่ได้ถูกอพยพก็จำเป็นต้องตามนโยบายและกลไกของการอพยพโพรเซสเหมือนกัน
3) ความแตกต่างกันของระบบ (Heterogeneous) การอพยพโพรเซสระหว่างระบบที่ต่างกันอาจทำให้เกิดปัญหาอันเนื่องมาจากคำสั่ง และสถาปัตยกรรมของซีพียูที่ต่างกัน
4) เคลื่อนย้ายหรือเปลี่ยนแปลงระบบ (Portability) กระทำได้ยากเนื่องจากการพัฒนาการอพยพโพรเซสในระดับระบบปฏิบัติการจะทำให้จำเป็นต้องมีการทำงานอยู่ภายใต้ระบบปฏิบัติการแบบเดียวกันเท่านั้น

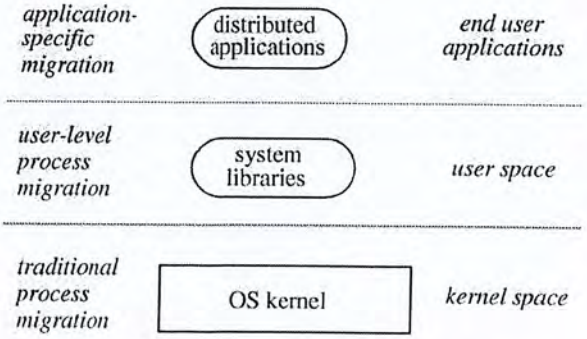
2.3.1 ระดับของการอพยพโพรเซส

เนื่องจากความซับซ้อนในการพัฒนาการอพยพโพรเซส รวมถึงความต้องการในแต่ละระบบปฏิบัติการที่ต่างกัน เป็นอุปสรรคอย่างมากในการที่จะพัฒนาการอพยพโพรเซสให้เกิดขึ้น จึงได้เกิดการแบ่งระดับของการอพยพโพรเซส เพื่อช่วยให้ง่ายต่อการพัฒนา โดยแต่ละระดับของการอพยพโพรเซสจะมีความแตกต่างกันในเรื่องของความซับซ้อนในการพัฒนา การนำกลับมาใช้ใหม่ และประสิทธิภาพการทำงาน

การอพยพโพรเซสในระดับแอปพลิเคชั่น (Application-Level) สามารถพัฒนาได้ง่าย ไม่ซับซ้อน แต่ส่งผลให้ต้องสูญเสียประสิทธิภาพในการทำงานบางส่วนไป การพัฒนาระบบจะเกิดขึ้นที่ตัวแอปพลิเคชั่นเอง ไม่เข้าไปยุ่งเกี่ยวกับภายในระบบปฏิบัติการ ทำให้สามารถทำงานได้กับทุกระบบ แต่การแก้ไข หรือปรับปรุงการอพยพโพรเซสจำเป็นต้องกระทำภายใต้แอปพลิเคชั่นเท่านั้น

ส่วนในระดับของผู้ใช้งาน (User-Level) สามารถพัฒนาได้ง่ายเช่นกัน ประสิทธิภาพในการทำงานจะดีกว่าระดับแอปพลิเคชั่น การทำงานส่วนใหญ่เน้นในด้านการกระจายโหลด (Load Distribution) สนับสนุนการทำงานประมวลผลที่ใช้ระยะเวลานานๆ สำหรับการพัฒนานั้น ไม่จำเป็นต้องยุ่งเกี่ยวกับเคอร์เนล แต่จะเน้นไปในส่วนของพื้นที่ผู้ใช้งาน (User-Space) และพื้นที่ของที่อยู่ (Address-Space) มากกว่า การอพยพโพรเซสในระดับนี้มักจะอพยพด้วยวิธีการคัดลอกส่วนของพื้นที่ผู้ใช้งาน และพื้นที่ที่อยู่ไปทั้งหมด แล้วจึงโอนถ่ายไปยังโหนดที่ต้องการเพื่อแตกข้อมูลเหล่านั้นไปทับพื้นที่บนโหนดปลายทาง ทั้งนี้อาจจะมีบางโพรเซสที่ไม่สามารถอพยพได้ อันเนื่องมาจากมีบางสถานะ หรือทรัพยากรในระดับผู้ใช้งานไม่สามารถถ่ายโอนไปสู่โหนดอื่นๆได้ จึงอาจต้องใช้การอพยพในระดับระบบปฏิบัติการเข้ามาช่วยต่อไป

สำหรับการอพยพโพรเซสในระดับระบบปฏิบัติการ (Kernel-Level) นั้น เป็นส่วนที่พัฒนาได้ยากที่สุด มีความซับซ้อนมากที่สุด เนื่องจากการพัฒนาในระดับระบบปฏิบัติการนั้นต้องเข้าใจสถาปัตยกรรม และการทำงานภายในระบบปฏิบัติการ เมื่อนำมาใช้งานแล้วการอพยพโพรเซสในระดับนี้จึงสามารถทำงานได้อย่างมีประสิทธิภาพมากที่สุด



รูป 2.2 ระดับของการพัฒนาการอพยพโพรเซส

2.3.2 แอปพลิเคชั่นที่ใช้หลักการอพยพโพรเซส

2.3.2.1 แอปพลิเคชั่นแบบขนาน

แอปพลิเคชั่นแบบขนานจะสามารถเริ่มทำงานได้ในโหนดหนึ่ง และอพยพโพรเซสในระดับแอปพลิเคชั่นเพื่อไปทำงานยังอีกโหนดหนึ่ง หรือใช้หลักของการกระจายโหลด (Load balancing) เพื่อย้ายการทำงานได้เช่นกัน ตัวอย่างเช่น เครื่องจักรเสมือนแบบขนาน (Parallel Virtual Machine, PVM) ซึ่งสนับสนุนการทำงานแบบขนาน และการติดต่อภายในระหว่างโพรเซส ทำให้สามารถอพยพโพรเซสในระดับชั้นแอปพลิเคชั่นได้ หรือ Migratory PVM ซึ่งเป็นส่วนขยายของ PVM ที่อนุญาตให้แต่ละการทำงานของแอปพลิเคชั่นแบบขนานสามารถอพยพไปยังโหนดต่างๆได้อย่างอิสระ เป็นต้น

2.3.2.2 แอปพลิเคชั่นที่มีการทำงานยาวนาน

แอปพลิเคชั่นที่มีการทำงานเป็นระยะเวลานาน มีโอกาสที่จะเกิดการผิดพลาดสูงขึ้น เช่นในกรณีระบบปิดตัวเองอัตโนมัติ หรือโหนดบางส่วนไม่ตอบสนองต่อการทำงาน จึงได้มีการนำหลักการของการอพยพโพรเซสเข้ามาช่วยในการแก้ไขปัญหาดังกล่าว ด้วยการทำให้แอปพลิเคชั่นนั้นเกิดการอพยพเมื่อระบบกำลังเกิดความเสียหาย นอกจากนี้การอพยพโพรเซสยังสนับสนุนการทำงานในระดับแอปพลิเคชั่นด้วยการสร้างกลไกที่เรียกว่า จุดตรวจสอบและรีสตาร์ท ในกรณีที่แอปพลิเคชั่นเกิดเหตุการณ์ผิดปกติขึ้นอีกด้วย

2.3.2.3 แอปพลิเคชั่นทั่วไปที่เกิดจากการใช้งานพร้อมกัน

ตัวอย่างเช่น ห้องแลปของที่กำลังถูกใช้งานจากนักศึกษาพร้อมกันทุกคน ซึ่งแต่ละคนก็มีงานที่จะต้องให้คอมพิวเตอร์ทำการประมวลผล หากเราสามารถนำหลักการอพยพโพรเซสเข้ามาประยุกต์ในการออกแบบห้องแลปนี้ได้ จะทำให้ประสิทธิภาพการทำงานของเครื่องคอมพิวเตอร์โดยรวมทำงานได้ดีขึ้น ทั้งนี้เนื่องจากว่า การทำงานของแต่ละเครื่องนั้นแตกต่างกันตามช่วงเวลา หากมีการประยุกต์ใช้การอพยพโพรเซสแบบพลวัต ที่สามารถกระจายโพรเซสให้กับโหนดหรือเครื่องอื่นๆที่มีการใช้งานน้อยกว่าได้อัตโนมัติ จะทำให้ปริมาณโหลดของแต่ละเครื่องลดลงอย่างมาก ถือว่าเป็นการประยุกต์ที่มีประโยชน์มากทีเดียว

2.3.2.4 แอปพลิเคชั่นทั่วไป

แอปพลิเคชั่นเหล่านี้มีเป้าหมายในการทำงานที่ต่างกัน ซึ่งบางแอปพลิเคชั่นอาจจะสามารถทำการอพยพตัวเองได้ หรืออาจถูกอพยพเนื่องจากสิทธิ์ในการทำงาน แต่ทั้งนี้ การตัดสินว่าแอปพลิเคชั่นไหนสามารถทำการอพยพโพรเซสได้จำเป็นจะต้องรู้ถึงรายละเอียดของแอปพลิเคชั่นนั้นก่อน เนื่องจากบางแอปพลิเคชั่น ที่มีการทำงานสั้นเกินไป ไม่เหมาะที่จะทำการอพยพโพรเซส เพราะการอพยพโพรเซสแต่ละครั้งนั้นมีค่าเสียหาย (Cost) ที่จำเป็นจำต้องจ่ายในการที่จะอพยพโพรเซส ทำให้เราต้องไตร่ตรองถึงผลได้ผลเสียให้ดีก่อนที่จะทำการอพยพโพรเซสในแต่ละครั้งเพื่อไม่ให้เกิดโอเวอร์เฮด (Overhead) จากการอพยพโพรเซสขึ้นนั่นเอง

2.3.2.5 แอปพลิเคชั่นที่มีการประยุกต์ใช้หลักการอพยพโพรเซสอยู่แล้ว

แอปพลิเคชั่นที่ถูกออกแบบให้สามารถใช้งานกับการอพยพโพรเซสจะสามารถทำการอพยพโพรเซสแบบพลวัตได้โดยไม่ต้องทำอะไรเพิ่มอีก หากเกิดเหตุการณ์ที่ผิดปกติ หรือมีการทำงานมากเกินไปในโหนดหนึ่งๆ โพรเซสจะถูกอพยพไปยังโหนดอื่นเพื่อกระจายการทำงานทันที

2.3.2.6 แอปพลิเคชั่นเครือข่าย

เป็นแอปพลิเคชั่นในรูปแบบล่าสุดที่ถูกคิดค้นขึ้นมาให้สามารถทำการอพยพโพรเซสได้ ตัวอย่างเช่น ออปเจ็คที่สามารถเคลื่อนที่ได้ ซึ่งการพัฒนาแอปพลิเคชั่นเครือข่ายตามหลักการของการอพยพโพรเซสจะใช้เทคนิคหลายอย่างเข้ามารวมกัน เพื่อให้สามารถทำงานได้อย่างมีประสิทธิภาพมากที่สุด

2.3.3 วิธีการที่ใช้ในการอพยพโพรเซส

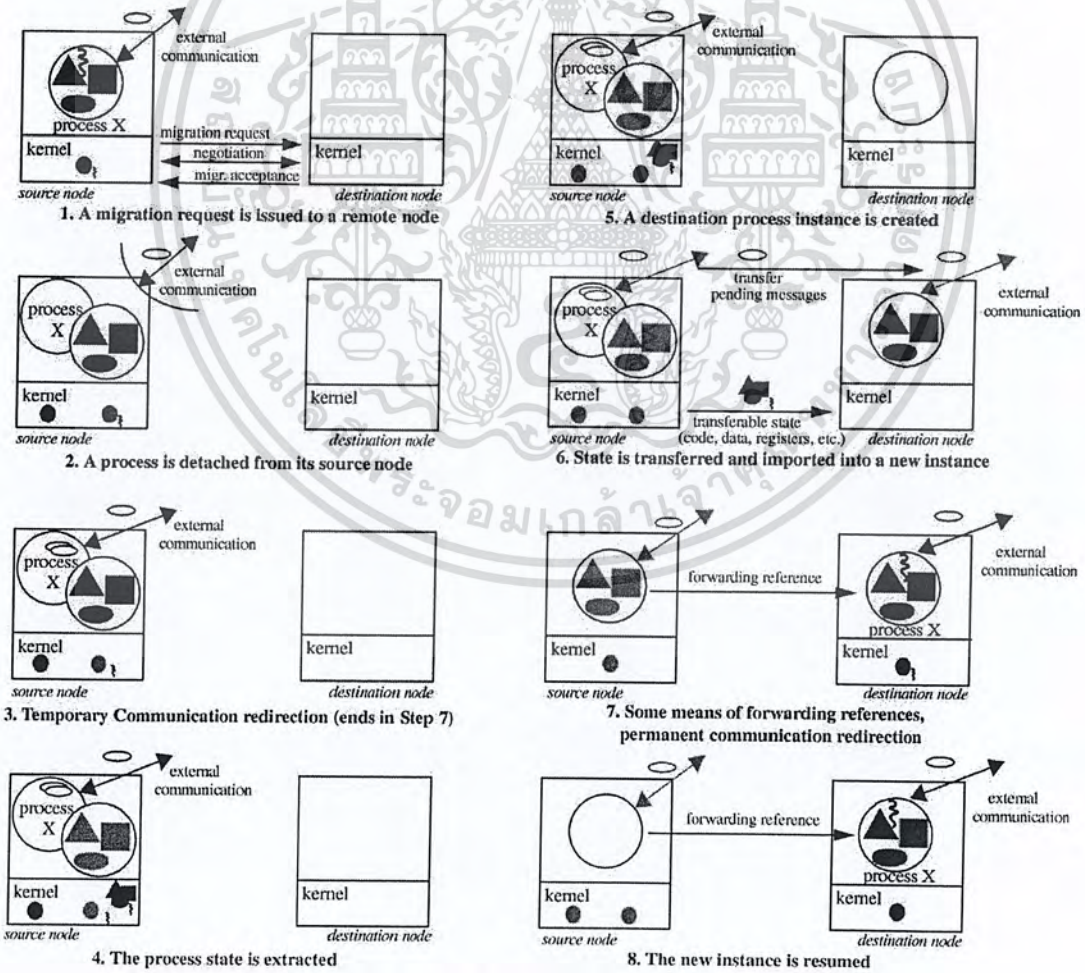
แม้ว่าวิธีการที่ใช้ในการพัฒนาการอพยพโพรเซสจะมีด้วยกันหลากหลาย แต่เราสามารถสรุปเป็นขั้นตอนได้ดังนี้

1) เกิดการร้องขอเพื่อทำการอพยพโพรเซสกับโหนดระยะไกล หลังจากที่เกิดการร้องขอแล้วจะมีการพิจารณาคำร้องนั้น การอพยพจะเกิดขึ้นก็ต่อเมื่อคำร้องขออพยพถูกอนุมัติ จากโหนดระยะไกลเท่านั้น
2) โพรเซสที่จะอพยพถูกถอดออกจากโหนดของมัน โดยการหยุดการทำงานของโพรเซส ประกาศให้โพรเซสอื่นๆรู้ว่าโพรเซสนั้นจะถูกอพยพ และเปลี่ยนทิศทางการติดต่อสื่อสารระหว่างโพรเซสนั้นชั่วคราว
3) การติดต่อสื่อสารจะถูกเปลี่ยนทิศทางโดยจะเกิดการเรียงคิวข้อความที่กำลังส่งเข้าหาโพรเซสที่กำลังจะอพยพ เพื่อให้สามารถส่งข้อความเหล่านี้ให้กับโพรเซสนั้นได้หลังจากที่การอพยพเสร็จสิ้นแล้ว โดยจะรอจนกว่าข้อความถูกส่งเข้ามาในคิวจนหมด ทำให้ช่องทางการติดต่อสื่อสารว่าง โพรเซสก็จะถูกอพยพออกจากโหนดปัจจุบันทันที
4) สถานะของโพรเซสถูกแยกออกเป็นส่วนๆ รวมทั้งส่วนของหน่วยความจำ ส่วนสถานะของโพรเซส ส่วนรีจิสเตอร์ ส่วนการติดต่อสื่อสารและส่วนของเคอร์เนลต่างๆ สำหรับส่วนการติดต่อสื่อสาร และส่วนของเคอร์เนลในบางระบบปฏิบัติการ อาจจะถูกจำกัดให้ใช้เฉพาะเครื่องเท่านั้น จึงทำให้ในบางกรณีส่วนเหล่านี้จะไม่ถูกส่งออกไปด้วย และส่วนของสถานะโพรเซสจะคงอยู่จนกว่าการอพยพส่วนอื่นๆจะเสร็จสิ้น
5) เกิดการสร้างโพรเซสที่โหนดปลายทาง เพื่อให้สามารถนำส่วนต่างๆที่ถูกถ่ายโอนจากโหนดก่อนเข้าสู่โหนดปลายทางได้ แต่การทำงานของโพรเซสนั้นจะยังไม่เริ่มทันที จะต้องรอจนกว่าส่วนต่างๆถูกอพยพมาเพียงพอต่อการทำงานแล้วเท่านั้น เมื่ออพยพมาจนครบทุกส่วนแล้ว โพรเซสนั้นก็จะเริ่มทำงานได้ตามปกติ

6) สถานะต่างๆถูกโอนถ่าย และถูกย้ายเข้าสู่โพรเซสตัวใหม่ในโหนดปลายทาง ไม่จำเป็นว่าทุกส่วนของโพรเซสเก่าจะต้องถูกอพยพมาทันที อาจมีบางส่วนที่ถูกอพยพมาภายหลังจากที่โพรเซสบนโหนดปลายทางเริ่มทำงานแล้วก็ได้

7) ในการอพยพโพรเซสนั้น อาจจำเป็นต้องมีแหล่งอ้างอิงของโพรเซสตัวเก่าเชื่อมโยงกับโพรเซสตัวใหม่บนโหนดปลายทาง เพื่อให้สามารถควบคุมโพรเซสนั้นได้มากยิ่งขึ้น สามารถทำได้โดยการลงทะเบียนจุดปัจจุบันของโพรเซสที่โหนดต้นทาง หรือการค้นหาโพรเซสที่อพยพแล้ว หรือทำการส่งต่อข้อความไปตามโหนดต่างๆ ซึ่งวิธีการเหล่านี้จะทำให้สามารถอพยพการติดต่อสื่อสารที่โหนดปลายทางได้นั่นเอง

8) โพรเซสใหม่จะสามารถสานต่อการทำงานได้หลังจากที่ได้รับสถานะต่างๆเพียงพอต่อการทำงาน เมื่อถึงขั้นตอนนี้ หมายถึงการอพยพโพรเซสได้เสร็จสิ้นลงแล้ว และหลังจากที่ทรัพยากรต่างๆถูกอพยพมาหมดแล้ว มันก็อาจจะถูกลบออกจากโหนดต้นทางได้ทันทีเช่นกัน



รูป 2.3 ขั้นตอนการอพยพโพรเซส

2.4 การจัดงานให้สมดุล (Load Balance)

การประมวลผลแบบขนานและกระจาย (Parallel and Distributed Computing) เป็นกระบวนการที่มีจุดประสงค์เพื่อแบ่งการคำนวณออกเป็นส่วนๆ โดยในแต่ละส่วนสามารถแก้โจทย์ปัญหาได้อย่างอิสระต่อกันภายในเครือข่ายเชื่อมต่อ (Interconnection Network) แต่การประมวลผลแบบนี้มักจะพบปัญหาเรื่องการจัดการงานที่ไม่สมดุล (Load Imbalance) เมื่อทำงานในสภาพแวดล้อมซึ่งประกอบด้วยเครื่องคอมพิวเตอร์ที่มีความสามารถต่างกัน ทำให้แอปพลิเคชั่นมีประสิทธิภาพลดลง แต่หากเครื่องเหล่านั้นสามารถแบ่งรับงานได้อย่างสมดุล จะช่วยเพิ่มระดับความน่าเชื่อถือและการใช้ประโยชน์ของระบบคอมพิวเตอร์ได้ ดังนั้นวิธีการในการจัดสรรทรัพยากรอย่างมีประสิทธิภาพ จึงมีผลต่อระบบประมวลผลแบบกระจาย กล่าวคือ การแบ่งงานควรจะกระจายปริมาณงานอย่างเท่าเทียมและยุติธรรม โดยหลักการกระจายงานนี้สามารถเกิดขึ้นได้ทั้งในระดับผู้ใช้ (User-Level) และในระดับระบบปฏิบัติการ (Kernel-Level)

การจัดสรรงานให้สมดุล (Load balancing) สามารถนิยามได้ว่า "หากมีงานซึ่งสามารถแบ่งออกได้เป็นงานย่อยและจะถูกคำนวณโดยใช้กลุ่มของคอมพิวเตอร์นั้น การจัดสรรงานให้สมดุลคือการจับกลุ่มงานย่อยอย่างเหมาะสม แล้วจัดสรรกลุ่มงานไปยังคอมพิวเตอร์เครื่องต่างๆ ซึ่งผลลัพธ์ที่ได้คือคอมพิวเตอร์แต่ละเครื่องนั้นจะมีขนาดปริมาณงานที่สมดุลกัน" การจัดสรรงานให้สมดุลไม่ได้หมายความว่า คอมพิวเตอร์ทุกเครื่องจะได้รับมอบหมายปริมาณงานที่เท่าเทียมกัน เนื่องจากเครื่องที่มีความสามารถสูงกว่าควรได้รับปริมาณงานที่มากกว่า จุดมุ่งหมายคือ การให้คอมพิวเตอร์สามารถทำงานที่ได้รับมอบหมายให้เสร็จสิ้นในเวลาเดียวกัน หรือใกล้เคียงกัน

หลักการของการจัดสรรงานให้สมดุลจะพยายามย้ายงานจากเครื่องที่มีปริมาณงานมากๆ ไปยังเครื่องที่มีปริมาณงานน้อยกว่า โดยการย้ายงานนั้นจะทำในรูปแบบที่พยายามจะลดเวลาตอบสนอง (Response time) ให้น้อยที่สุดและปรับประสิทธิภาพโดยรวมของระบบให้เหมาะสมที่สุด ดังนั้นการจัดสรรงานให้สมดุลจึงเป็นปัญหาของการจัดลำดับการทำงานหรือการจัดสรรและจัดการทรัพยากร

การจัดสรรงานให้สมดุลนั้นสามารถทำได้ทั้งก่อนเริ่มการประมวลผลและในระหว่างประมวลผล

ในการจัดสรรงานให้สมดุลก่อนการประมวลผล หรือที่เรียกว่าแบบสถิต (Static Load Balancing) ปริมาณงานจะถูกปรับให้สมดุลเมื่อเริ่มต้นการประมวลผล วิธีนี้ต้องอาศัยความรู้เกี่ยวกับหลักการทำงานของโปรแกรมประยุกต์นั้นๆ และสถานะต่างๆ ของระบบ ซึ่งไม่สามารถหาได้เสมอไป ดังนั้นการออกแบบวิธีการนี้จึงมีข้อจำกัดและเหมะสมกับโปรแกรมประยุกต์บางประเภทเท่านั้น

ส่วนการจัดสรรงานให้สมดุลระหว่างการประมวลผล หรือที่เรียกว่า แบบพลวัต (Dynamic Load Balancing) ปริมาณงานในแต่ละเครื่องสามารถเปลี่ยนแปลงไปได้ระหว่างการประมวลผล ซึ่ง

จะใช้ความรู้เกี่ยวกับระบบและโปรแกรมประยุกต์น้อยมาก จึงทำให้วิธีการนี้มีประสิทธิภาพดีกว่าการจัดสรรงานให้สมดุลแบบสถิต อย่างไรก็ตามวิธีการนี้ทำได้ค่อนข้างยาก เนื่องจากต้องมีการโยกย้ายงานหลังการประมวลผลได้เริ่มต้นแล้ว จึงทำให้เกิดโอเวอร์เฮดในการคำนวณโหลดและการสื่อสาร

2.4.1 การจัดสรรงานให้สมดุลแบบสถิต (Static Load Balancing)

เทคนิคในการจัดสรรงานให้สมดุลแบบสถิตมักเกี่ยวข้องกับการจัดสรรงานย่อยไปยังหน่วยประมวลผลเพื่อลดเวลาในการประมวลผลโดยรวม และเวลาในการติดต่อสื่อสารกันระหว่างหน่วยประมวลผล ส่วนต่อไปนี้จะเป็นการยกตัวอย่างบางเทคนิคโดยสังเขป

2.4.1.1 การใช้ทฤษฎีกราฟ (The Graph Theoretic Approach)

ในวิธีการนี้ โมดูลย่อยๆของโปรแกรมจะถูกแสดงด้วยจุดบนกราฟที่มีทิศทางแน่นอน โดยเส้นกราฟจะแสดงการไหลของข้อมูลระหว่างโมดูลต่างๆ น้ำหนักของเส้นกราฟจะแสดงถึงเวลาที่ต้องใช้ และโอเวอร์เฮดที่เกิดขึ้นในการส่งข้อมูล ซึ่งในการไหลของข้อมูลผ่านระบบเครือข่ายที่เหมาะสมนั้นควรเป็นไปตามสมการดังต่อไปนี้

$$\sum flow\_in = \sum flow\_out \qquad (2.1)$$

โดยที่ *flow_in* และ *flow_out* นั้นไม่เป็นค่าลบ วิธีการก็คือ ผู้พัฒนาโปรแกรมต้องหาค่าการไหลของข้อมูลที่มีค่ารวมน้อยที่สุดจากค่าการไหลของข้อมูลทั้งหมดที่เป็นไปได้ โดยอาศัยการวาดกราฟที่มีทิศทาง ซึ่งจุดในกราฟหนึ่งจุดจะแทนหน่วยประมวลผลหนึ่งหน่วย หรือแทนโมดูลหนึ่งโมดูลในโปรแกรม จากนั้นให้เส้นที่ลากระหว่างจุดที่แสดงโมดูลของโปรแกรมและจุดที่แทนหน่วยประมวลผลแสดงถึงเวลาและโอเวอร์เฮดในการประมวลผลของโมดูลบนหน่วยประมวลผลนั้นๆ ผู้พัฒนาโปรแกรมจะสามารถหางานที่เหมาะสมได้จากการคำนวณหาน้ำหนักที่น้อยที่สุดจากการตัดองค์ประกอบย่อยตามแนวตั้งหรือแนวขวาง (Cut-Set) ซึ่งการคำนวณวิธีนี้ในระบบเครือข่ายที่มีขนาดใหญ่นั้นจะใช้เวลามาก เนื่องจากมีความซับซ้อนของเวลาสูงเป็น $O(mn^2)$ เมื่อ m เป็นจำนวนของโมดูลและ n เป็นจำนวนของหน่วยประมวลผล

2.4.1.2 วิธีการหาคำตอบแบบลองผิดลองถูก (Heuristic Approach)

วิธีการนี้มีหลักการคือ การลองผิดลองถูกเพื่อค้นหาหลายๆคำตอบ แล้วเลือกคำตอบที่น่าจะเหมาะสมที่สุดมาใช้งาน การคำนวณตามหลักการดังกล่าวนั้นง่ายและใช้เวลาน้อย วิธีการนี้จะตัดสินใจเลือกคำตอบโดยอาศัยตัวแปรที่มีความสัมพันธ์ทางอ้อมกับประสิทธิภาพของระบบ ซึ่งตัวแปรเหล่านี้สามารถคำนวณและตรวจสอบได้ง่าย ยกตัวอย่างของวิธีการหาคำตอบแบบลองผิดลองถูก เช่น

1) ลดการติดต่อสื่อสาร (Minimize Communication) โดยการหาโมดูลที่มีการติดต่อกันระหว่างโมดูลมากที่สุด แล้วนำโมดูลเหล่านั้นมาไว้ในหน่วยประมวลผลเดียวกัน

2) ลดเวลาที่ใช้ในการประมวลผล (Minimize Execution Time) โดยจัดลำดับความเร็วของหน่วยประมวลผลทั้งหมด ซึ่งหน่วยประมวลผลที่มีความเร็วมากกว่าจะได้รับงานมากกว่า

2.4.2 การจัดสรรงานให้สมดุลแบบพลวัต (Dynamic Load Balancing)

หากปริมาณงานสามารถเปลี่ยนแปลงได้ตลอดช่วงเวลาของการคำนวณและไม่สามารถคาดการณ์ล่วงหน้าได้ การจัดกลุ่มของงานนั้นก็จะเปลี่ยนแปลงอยู่ตลอดเวลาในระหว่างการประมวลผล ซึ่งอาจกล่าวได้ว่า การจัดกลุ่มของงานนั้นต้องเกิดขึ้นพร้อมๆกับการประมวลผลนั่นเอง ทั้งนี้ปัจจัยที่มีผลต่อการจัดสรรงานให้สมดุลนั้นมีดังต่อไปนี้

1) ปริมาณงานของระบบ (System Load) สถานะของระบบสามารถหาค่าได้จากการประเมินปริมาณงานในแต่ละหน่วยประมวลผล

2) เงื่อนไขเกี่ยวกับปริมาณข้อมูลเข้าออกของเครือข่าย (Network Traffic Condition) สถานะของเครือข่ายสามารถหาได้จากการทดลองหรือคาดการณ์จากข้อมูลเครือข่ายบางส่วนที่มีอยู่

3) ลักษณะเฉพาะของงาน (Characteristics of Tasks) จะเกี่ยวข้องกับขนาดของงาน ซึ่งขึ้นอยู่กับเวลาที่ใช้ในการประมวลผล เวลาในการเคลื่อนย้ายงานและรายละเอียดของงานนั้นๆว่าเป็นโพรเซสประเภทที่มีการใช้ซีพียูหรืออินพุต เอาต์พุตสูงกว่าเมื่อนำเวลาที่ใช้มาเปรียบเทียบกัน (CPU Bound, I/O Bound)

การคาดการณ์เกี่ยวกับการประมวลผลของงานที่ดีจะช่วยในการตัดสินใจจัดสรรงานให้สมดุล ซึ่งการคาดการณ์เกี่ยวกับเวลาที่ใช้ในการประมวลผลของโปรแกรมสามารถทำได้โดยการสังเกตการณ์ทำงานในครั้งก่อนๆของโปรแกรมประยุกต์ที่มีลักษณะคล้ายกัน หรือวิเคราะห์จากเทคนิคการทำแบบจำลอง

โดยทั่วไปการจัดสรรงานให้สมดุลแบบพลวัตจะประกอบด้วยส่วนสำคัญหลักๆ 3 ส่วน คือ เกณฑ์การวัดปริมาณงานในหน่วยประมวลผล วิธีการแลกเปลี่ยนข้อมูลสถานะของระบบ และวิธีการโยกย้ายงาน

2.4.2.1 เกณฑ์ของการวัดปริมาณงานในหน่วยประมวลผล (Processor Load Measurement)

เกณฑ์การวัดจะแสดงให้เห็นถึงลักษณะโดยรวมของงานนั้นๆ ซึ่งการหาค่าเหล่านี้มักจะต้องทำบ่อยครั้ง ดังนั้นจึงควรทำอย่างมีประสิทธิภาพเพื่อไม่ให้เป็นการเพิ่มโอเวอร์เฮดให้การ

ประมวลผลโดยไม่จำเป็น โดยในบางเกณฑ์นั้นจะมีอยู่แล้วในระบบปฏิบัติการยูนิกซ์ (UNIX) ตัวอย่างเกณฑ์การวัดปริมาณงานในหน่วยประมวลผลมีดังนี้

1) จำนวนงานในคิวที่กำลังรอการประมวลผล

2) อัตราการเรียกใช้งานฟังก์ชั่นที่เป็นของระบบปฏิบัติการ

3) อัตราการเกิดคอนเท็กซ์สวิตช์ของซีพียู

4) ปริมาณเวลาว่างของซีพียู

5) ขนาดของหน่วยความจำที่ยังไม่ถูกใช้งาน

6) ค่าเฉลี่ยของปริมาณงานในเวลา x นาที

แต่ละเกณฑ์ดังกล่าวจะมีความสำคัญไม่เท่ากัน ขึ้นอยู่กับลักษณะของโปรแกรมประยุกต์นั้นๆ เช่น หากว่างานทั้งหมดในโปรแกรมประยุกต์สามารถคำนวณเสร็จภายในเวลาน้อยกว่า 1 นาทีแล้ว ค่าเฉลี่ยของปริมาณงานในเวลา 1 นาทีก็จะไม่มีประโยชน์นัก หรือหากโปรแกรมมีการสลับข้อมูลระหว่างหน่วยความจำในลำดับชั้นย่อยๆ (Memory Paging) ขนาดของหน่วยความจำที่ยังไม่ถูกใช้งานก็จะเป็นตัวบ่งชี้ที่ดีสำหรับปริมาณงานในหน่วยประมวลผลนั้น

2.4.2.2 วิธีการแลกเปลี่ยนข้อมูลสถานะของระบบ (System Information Exchange Policy)

วิธีการจะเป็นตัวระบุว่าหน่วยประมวลผลแต่ละหน่วยควรจะเริ่มหาและเก็บค่าสถานะระบบของตัวเอง ณ เวลาใดบ้าง นอกจากนี้ยังเป็นตัวแสดงว่าแต่ละหน่วยประมวลผลควรส่งข้อมูลเกี่ยวกับสถานะของตัวเองไปยังหน่วยประมวลผลอื่นๆในกลุ่มในเวลาใดบ้างด้วย

หลักการในการเลือกความถี่ของการหาและส่งข้อมูลสถานะจะอ้างอิงจากความสัมพันธ์ระหว่างสถานะของระบบเครือข่ายและการหาค่าความเปลี่ยนแปลงของปริมาณงานในแต่ละจุด โดยหากงานมีการรับส่งข้อมูลบนเครือข่ายในปริมาณมากแล้ว การแลกเปลี่ยนข้อมูลระบบนั้นควรถูกหยุดไว้ก่อน ซึ่งสถานะของระบบเครือข่ายนั้นสามารถคาดการณ์ได้จากการส่งข้อความสั้นๆไปกลับระหว่างหน่วยประมวลผล 2 หน่วย เพื่อหาค่าความล่าช้าของเครือข่าย (Network Delay) หรือใช้ข้อมูลของปริมาณการรับส่งข้อมูลในอดีตมาประมาณการรับส่งข้อมูลในอนาคตก็ได้ ตัวอย่างข้อมูลระบบหลายชนิดที่เป็นประโยชน์และสามารถถูกแลกเปลี่ยนได้ เช่น สถานะของหน่วยประมวลผล (สถานะปกติ – idle, สถานะกำลังยุ่ง – busy, สถานะที่ปริมาณงานน้อย -–low load, สถานะที่มีปริมาณงานมาก – heavy load) ตัวอย่างของวิธีการใช้ข้อมูลสถานะเพื่อการโยกย้ายงานมีดังต่อไปนี้

2.4.2.2.1 วิธีการที่มีขอบเขตจำกัด (The limited approach)

เมื่อมีหน่วยประมวลผลหนึ่งถูกใช้งานมากจนเกินไป จะเกิดการสุ่มหน่วยประมวลผลขึ้นมาจำนวนหนึ่งเพื่อตรวจสอบสถานะ โดยการแลกเปลี่ยนข้อมูลระบบกัน จากนั้นจะทำการเลือกหน่วยประมวลผลจากกลุ่มที่สุ่มขึ้นมาเพื่อรับแบ่งงานไปทำเพิ่มเติม

2.4.2.2.2 การจับคู่ (The pairing approach)

หน่วยประมวลผลจะส่งปริมาณงานของตนไปยังหน่วยประมวลผลที่อยู่ในตำแหน่งใกล้เคียงกัน แล้วทำการจับคู่กับหน่วยประมวลผลตัวที่มีปริมาณงานน้อยที่สุด เพื่อแบ่งงานให้ไปทำเพิ่มเติม

2.4.2.2.3 การใช้ตัวบ่งชี้ปริมาณงาน (The load vector approach)

ตัวบ่งชี้ปริมาณงานของกลุ่มของหน่วยประมวลผลจะถูกเก็บรักษาไว้ที่ส่วนกลาง และจะนำมาใช้เมื่อต้องการจัดสรรงานให้สมดุล กล่าวคือ ความแตกต่างของปริมาณงานที่เก็บไว้จะถูกนำมาเปรียบเทียบเพื่อประกอบการตัดสินใจเลือกหน่วยประมวลผลที่จะส่งและรับงาน

2.4.2.2.4 วิธีการกระจายข้อมูลสถานะ (The broadcast approach)

วิธีการนี้จะมีโพรเซสตัวหนึ่งคอยทำการตรวจสอบค่าเฉลี่ยของปริมาณงานในระบบปฏิบัติการยูนิกซ์ของหน่วยประมวลผลแต่ละตัวอยู่ตลอดเวลา เช่น ทุก 1 นาที หรือทุก 5 นาที เมื่อใดที่หน่วยประมวลผลสามารถรับงานได้มากขึ้น โพรเซสนี้ก็จะกระจายข้อมูลออกไปให้หน่วยประมวลผลอื่นๆรับราบ

2.4.2.2.5 ปริมาณงานโดยรวมของระบบ (The global system load approach)

ปริมาณงานโดยรวมของระบบจะถูกกระจายออกไปยังทุกหน่วยประมวลผล จากนั้นแต่ละหน่วยประมวลผลจะทำการเปรียบเทียบปริมาณงานของตัวเองกับปริมาณงานโดยรวมของระบบ เมื่อปริมาณงานเกิดความแตกต่างกันมาก ก็จะเริ่มมีการจัดสรรงานให้สมดุลโดยการโยกย้ายงาน

2.4.3 วิธีการโยกย้ายงาน (Transfer Policy)

การตัดสินใจในการโยกย้ายงานจะขึ้นอยู่กับสภาพแวดล้อมของระบบ ณ. ขณะนั้น การโยกย้ายงานจะเกิดขึ้นเมื่อหน่วยประมวลผลมีงานมากจนเกินความสามารถที่จะทำให้เสร็จตามเวลา หรือมีงานน้อยจนทำให้ประสิทธิภาพการใช้งานตกลง นอกจากนี้ การตัดสินใจโยกย้ายงานต้องเลือกว่าหน่วยประมวลผลใดจะส่งและรับงานอีกด้วย โดยทั่วไปแล้วหากระบบตัดสินให้มีการโยกย้ายงานเกิดขึ้น งานที่ถูกเลือกให้ย้ายหน่วยประมวลผลควรจะเป็นงานที่เพิ่งเริ่มประมวลผลได้ไม่นาน หลักการเลือกงานทั่วไปสรุปได้ดังต่อไปนี้

1) การเคลื่อนย้ายงานที่อยู่ในสถานะ “หยุดทำงาน” (Blocked) อยู่อาจไม่มีประโยชน์มากนัก เนื่องจากการเคลื่อนย้ายงานจะไม่ส่งผลกระทบใดๆ ต่อปริมาณงานในหน่วยประมวลผลนั้นๆ
2) การเคลื่อนย้ายงานที่ถูกเลือกจากคิวให้ประมวลผลอยู่ในขณะนั้น (Scheduled process) อาจส่งผลให้เกิดโอเวอร์เฮดในการย้ายสูงกว่าปกติ

3) การย้ายงานที่มีขนาดเล็กทำให้เกิดปริมาณการติดต่อสื่อสารบนเครือข่ายเพียงเล็กน้อยเท่านั้น

4) การย้ายงานที่ยังเหลือเวลาในการประมวลผลอีกมาก จะเกิดประโยชน์ในระยะยาวมากที่สุด

5) การเคลื่อนย้ายงานที่ได้มีการติดต่อสื่อสารกับงานอื่นๆที่ประมวลผลอยู่บนหน่วยประมวลผลปลายทางอยู่แล้ว จะช่วยลดปริมาณการรับ – ส่งบนเครือข่ายได้

วิธีการเลือกหน่วยประมวลผลต้นทางและปลายทางสำหรับการโยกย้ายงานที่เหมาะสมนั้น อาจเกิดจากผู้ส่งเป็นผู้ริเริ่มการขอย้ายงาน (Sender – Initiated) หรือผู้รับเป็นผู้ริเริ่ม (Receiver – Initiated) ก็ได้ ซึ่งโดยปกติแล้วในกรณีที่ระบบไม่ได้มีปริมาณงานมากนัก การเคลื่อนย้ายงานโดยผู้ส่งเป็นผู้ริเริ่มจะทำงานได้ดีกว่า ในขณะที่การเคลื่อนย้ายงานแบบผู้รับเป็นผู้เริ่มนั้นจะเหมาะกับระบบประมวลผลที่มีปริมาณงานโดยรวมจำนวนมาก

วิธีการย้ายงานแบบผู้ส่งเป็นผู้เริ่มจะเริ่มจากการที่หน่วยประมวลผลที่มีงานล้นมือทำการเลือกหน่วยประมวลผลปลายทางสำหรับการย้ายงาน ซึ่งกรรมวิธีการเลือกหน่วยประมวลผลปลายทางสำหรับการย้ายงาน ซึ่งกรรมวิธีการเลือกสามารถทำได้โดยการสุ่ม จากนั้นปริมาณงานบนหน่วยประมวลผลที่ถูกสุ่มเลือกจะถูกตรวจสอบ ซึ่งหากปริมาณงานมีค่าน้อยกว่าค่าที่กำหนดไว้ งานจำนวนหนึ่งก็จะถูกย้ายมาประมวลผลยังหน่วยประมวลผลดังกล่าว

ส่วนวิธีการย้ายงานแบบผู้รับเป็นผู้เริ่มจะเริ่มจากหน่วยประมวลผลที่มีปริมาณงานน้อยกว่าค่าที่กำหนดไว้ทำการโพล (poll) ไปยังหน่วยประมวลผลอื่นๆแบบสุ่มเลือกเพื่อขอรับงานเพิ่ม หรือทำการกระจายข้อความบอกสถานะของตัวเองออกไปยังหน่วยประมวลผลทั้งหมดในกลุ่ม

2.4.4 การย้ายงาน (Task Migration)

หลังจากขั้นตอนของการคัดเลือกงาน ระบบจะจัดเตรียมวิธีการในการย้ายงานจริงจากคอมพิวเตอร์เครื่องหนึ่งไปยังคอมพิวเตอร์อีกเครื่อง โดยการเคลื่อนย้ายงานนั้นจะต้องรักษาสถานะของการประมวลผลต่างๆไว้ ในขณะเดียวกันก็ควรเก็บเส้นทางการติดต่อสื่อสารที่เกิดขึ้นก่อนการเคลื่อนย้ายเอาไว้ด้วย ทั้งนี้ เมื่อการเคลื่อนย้ายเสร็จสิ้น งานต่างๆจะได้สามารถเริ่มการประมวลผลต่อจากจุดเดิมได้เลย โดยไม่ต้องเริ่มต้นใหม่ ซึ่งการเคลื่อนย้ายงานจำเป็นต้องอาศัยการเพิ่มโค้ดในโปรแกรมประยุกต์ โดยเฉพาะอย่างยิ่งการเคลื่อนย้ายข้อมูลที่มีโครงสร้างซับซ้อน กล่าวคือ ผู้พัฒนาโปรแกรมจะต้องเตรียมฟังก์ชั่นการบรรจุ และการแกะออก (Pack and Unpack) ของข้อมูลเพื่อจัดการกับการเคลื่อนย้ายข้อมูลของโปรแกรมประยุกต์เหล่านั้น

จากการศึกษาแสดงให้เห็นว่า การลดขนาดของปริมาณงานที่ต้องเคลื่อนย้ายนั้นมีความสำคัญมากกว่าการลดจำนวนของงานที่ต้องทำการเคลื่อนย้าย

หลักการต่างๆในข้างต้น สามารถนำมาใช้ประกอบการออกแบบ และพัฒนาโปรแกรมที่ใช้เพื่อจัดสรรงานให้สมดุลได้ ซึ่งโปรแกรมที่ออกแบบควรมีลักษณะดังต่อไปนี้

1) มีประสิทธิภาพ (Efficiency) การจัดสรรงานจะต้องได้ประโยชน์มากกว่าโอเวอร์เฮดที่เกิดขึ้น

2) มีเสถียรภาพ (Stability) งานต้องไม่ถูกเคลื่อนย้ายตลอดเวลา

3) มีความสามารถในการเพิ่มจำนวนหน่วยประมวลผล (Scalability) โดยที่การเพิ่มจำนวนหน่วยประมวลผลที่ใช้งานได้ ไม่ควรมีผลกระทบต่อประสิทธิภาพโดยรวมของระบบ

4) สามารถจัดรูปแบบได้ (Configurability) โปรแกรมที่ใช้เพื่อจัดสรรงานให้สมดุลควรมีโครงสร้างที่สามารถปรับให้ตรงกับความต้องการของโปรแกรมประยุกต์ประเภทต่างๆได้

5) มีความครอบคลุม (Generality) โปรแกรมจัดสรรงานให้สมดุลควรมีความหลากหลายสำหรับใช้กับโปรแกรมประยุกต์ที่แตกต่างกัน

6) สามารถรองรับสิ่งที่ต่างกันได้ (Heterogeneity) โปรแกรมจัดสรรงานให้สมดุลควรทำงานได้กับหน่วยประมวลผลประเภทต่างๆ ซึ่งมีความสามารถที่แตกต่างกันได้ รวมทั้งสามารถทำงานกับระบบปฏิบัติการและสถาปัตยกรรมของเครื่องที่ต่างกันอีกด้วย

7) ซ่อนความซับซ้อนจากผู้ใช้ (Transparent) ผู้ใช้ไม่ควรต้องทำการปรับเปลี่ยนโค้ดจำนวนมากเพื่อให้สามารถใช้โปรแกรมจัดสรรงานให้สมดุลได้

2.5 ระบบจดหมายอิเล็กทรอนิกส์ (E-Mail System)

ในอดีตเทคโนโลยียังไม่พัฒนามากนัก การติดต่อที่สะดวกก็คงเป็นโทรศัพท์ ถ้าพูดถึงเรื่องการส่งเอกสารก็คงมีเครื่องโทรสาร แต่เครื่องโทรสารคงไม่มีใช้กันตามบ้านทั่วไป ดังนั้นระบบไปรษณีย์จึงเป็นหนทางหนึ่งที่เราใช้ติดต่อกัน ไม่ว่าจะเป็นโทรเลข พัสดุภัณฑ์ ธนาณัติ และที่ขาดไม่ได้คงเป็นจดหมาย ซึ่งโดยปกติจดหมายที่ส่งแบบธรรมดาภายในจังหวัดก็คงจะวันเดียว แต่ถ้าเป็นต่างจังหวัด และยิ่งเป็นอำเภอที่ไกลจากตัวเมืองก็ยิ่งต้องใช้เวลามากขึ้นไปด้วย ถ้าให้เร็วขึ้นก็มีบริการ EMS ซึ่งราคาก็จะแพงตามน้ำหนัก

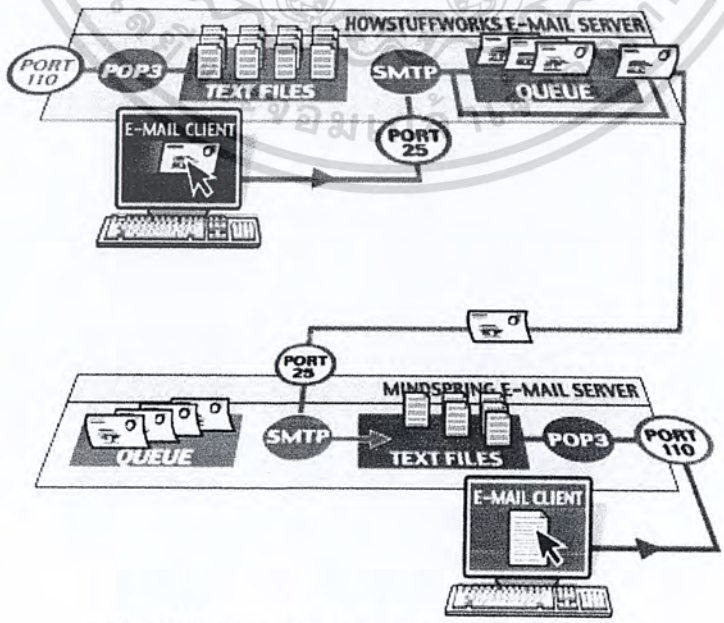
ในปัจจุบันเมื่ออินเตอร์เน็ตเข้ามามีบทบาทมากขึ้น การติดต่อกันจึงมีช่องทางมากขึ้นสะดวกขึ้น จากรูปของจดหมายที่เป็นแบบกระดาษ ก็ผันเปลี่ยนมาอยู่ในรูปแบบของจดหมายอิเล็กทรอนิกส์ หรือที่รู้จักกันคือ อีเมล์ (E-Mail) เป็นการติดต่อสื่อสารที่เป็นที่นิยมมากขึ้น ขอเพียงมีไฟฟ้า โทรศัพท์ และคอมพิวเตอร์ เราก็สามารถติดต่อกันไม่ว่าอยู่ที่มุมไหนของโลกเพียงไม่กี่นาทีเท่านั้น

อีเมล์หรือจดหมายอิเล็กทรอนิกส์ ได้มีการใช้งานมานานแล้ว ตั้งแต่ยุคของเครื่องเมนเฟรมหรือมินิคอมพิวเตอร์ ซึ่งไอบีเอ็มได้พัฒนาระบบที่เรียกว่า PROFS (Professional Office System) ออกมา

ใช้งาน นอกจากนี้ก็มีบนระบบยูนิกซ์ ต่อมาหลายค่ายก็ได้พัฒนาระบบอีเมล์ของตนขึ้นมา โดยส่วนใหญ่จะเป็นองค์ประกอบในแอปพลิเคชั่นที่ทำงานบนระบบเครือข่าย เช่น ไมโครซอฟต์ เมล์ (Microsoft Mail) ของบริษัทไมโครซอฟต์ ซีซีเมล์ (CC Mail) ของโลตัส (LOTUS) เป็นต้น ซึ่งต่างก็ได้ใช้เทคโนโลยีของตนเองและเป็นระบบปิด ดังนั้นการส่งเมล์ไปยังผู้ใช้ที่มีระบบเมล์คนละค่ายกันจึงเป็นเรื่องที่ยุ่งยาก ในยุคต่อมามีระบบเครือข่ายทั้ง LAN และ WAN ต่างมีมาตรฐานและเป็นระบบเปิด (Open System) มากขึ้น ก็ได้มีการปรับเปลี่ยนการทำงานของระบบเมล์มาเป็นแบบ Client – Server ที่เป็นพื้นฐานแบบที่ใช้กันในระบบยูนิกซ์ และมีการพัฒนาระบบเมล์ ได้ทั้งโดยการผ่านระบบ LAN หรือใช้โมเด็มเขามาจาก WAN ทำให้ผู้ใช้จะไม่เห็นไฟล์ในฮาร์ดดิสก์บนเซิร์ฟเวอร์เลย ดังนั้นความปลอดภัยจึงมีมากขึ้น จนในปัจจุบัน ได้พัฒนาขึ้นมาเป็นระบบเวิร์คโฟล (Workflow) ที่ใช้อีเมล์เป็นพื้นฐาน

2.5.1 ลักษณะการทำงานของระบบรับ – ส่งเมล์

ผู้ส่ง (Sender) เริ่มเขียนจดหมาย จากนั้นส่งผ่านโปรโตคอลไปยังเครื่องเมล์เซิร์ฟเวอร์ต้นทาง จากนั้นเมล์เซิร์ฟเวอร์จะส่งไปยังเครื่องที่เป็นรีเลย์เครื่องเมล์เซิร์ฟเวอร์ต้นทาง เนื่องจากเครื่องรีเลย์นี้สามารถติดต่อกับเมล์เซิร์ฟเวอร์เครื่องอื่นๆที่อยู่ภายนอกได้ (เครื่องเมล์เซิร์ฟเวอร์และเครื่องรีเลย์อาจเป็นเครื่องเดียวกันก็ได้) เมื่อเครื่องรีเลย์ของต้นทางได้รับอีเมล์ ก็จะทำการติดต่อกับเครื่องรีเลย์ปลายทางเพื่อส่งอีเมล์ฉบับนี้ไป และถ้าหากเครื่องรีเลย์ปลายทางและเมล์เซิร์ฟเวอร์ปลายทางเป็นเครื่องเดียวกัน เมื่อเมล์ไปถึงเครื่องนั้นก็จะถือว่าจบการส่งเมล์ทันที เมื่อผู้รับ (Receiver) ทำการเช็คเมล์ หรืออ่านเมล์ ก็จำเป็นต้องมีการติดต่อกับเมล์เซิร์ฟเวอร์ด้วยวิธีต่างๆ เพื่อนำเมล์ฉบับนี้มาอ่าน



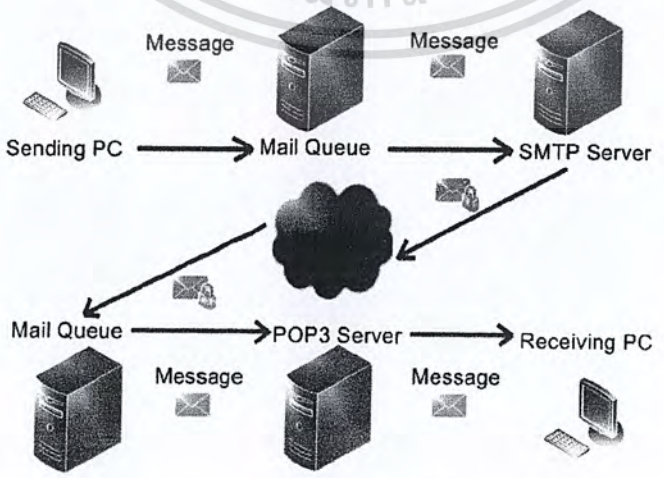
รูป 2.4 ลักษณะการทำงานของระบบรับ – ส่งเมล์

2.5.2 สถาปัตยกรรมของระบบเมล์

ทีซีพีไอพี มีโปรโตคอลสนับสนุนการรับส่งเมล์หลายโปรโตคอล แต่โปรโตคอลที่นิยมใช้ในอินเตอร์เน็ตคือ SMTP (Simple Mail Transfer Protocol) หน้าที่ของ SMTP คือกำหนดกรรมวิธีและแบบแผนการนำส่งข้อความระหว่างผู้รับและผู้ส่ง โดย SMTP อาศัยโปรโตคอลทีซีพีเพื่อลำเลียงจดหมายผ่านพอร์ต 25 ระบบเมล์ที่ใช้ในทีซีพีไอพี มีองค์ประกอบสองส่วนคือ ยูสเซอร์เอเจนท์ (User – Agent, UA) หรืออีกชื่อคือ เมล์ยูสเซอร์เอเจนท์ (Mail User Agent, MUA) และเมล์ทรานสเฟอร์เอเจนท์ (Mail Transfer Agent, MTA)

UA ทำหน้าที่ในการติดต่อกับผู้ใช้เพื่อรับและส่งเมล์ ซึ่งรูปแบบของการติดต่อมี 3 แบบดังนี้

1) การติดต่อโดยตรง หรือทำงานบนเครื่องที่เก็บกล่องจดหมาย ซึ่งโปรแกรมที่ใช้ในการรับส่งเมล์ที่นิยมกันบนลีนุกซ์ ก็เช่น /bin/mail mailx หรือ pine เป็นต้น โดยการใช้งานจริงอาจจะด้วยการ เทลเน็ต (Telnet) จากเครื่องคอมพิวเตอร์เข้าไปยังเครื่องที่เป็นเมล์เซิร์ฟเวอร์ แล้วใช้งานโปรแกรมดังกล่าวบนเมล์เซิร์ฟเวอร์
2) การทำงานแบบไคลเอนต์ เซิร์ฟเวอร์โดยที่เครื่องเป็นเมล์ไคลเอนต์ (Mail Client) จะติดต่อกับเครื่องเมล์เซิร์ฟเวอร์ (Mail Server) ผ่านโปรโตคอลสำหรับการจัดการโดยเฉพาะ เช่น POP3 (Post Office Protocol version 3) หรือ IMAP 4 (Internet Mail Access Protocol version 4) เพื่อให้ดึงจดหมายจากกล่องจดหมายบนเซิร์ฟเวอร์ไปอ่าน ซึ่งโปรแกรมที่นิยมใช้งานเป็นเมล์ไคลเอนต์ เช่น Microsoft Outlook เป็นต้น
3) การทำงานแบบเว็บเมล์ เป็นการติดต่อระหว่างเว็บเซิร์ฟเวอร์ที่มีโปรแกรมเว็บเมล์ติดตั้งอยู่กับเมล์เซิร์ฟเวอร์ โปรโตคอลที่นิยมใช้กันจะเป็น IMAP ซึ่งเมล์เซิร์ฟเวอร์กับเว็บเซิร์ฟเวอร์อาจเป็นเซิร์ฟเวอร์เดียวกันหรือต่างกันก็ได้ เช่น hotmail.com เป็นต้น



รูป 2.5 การทำงานของเมล์เซิร์ฟเวอร์

Mail Transport Agent (MTA) คือส่วนที่ทำหน้าที่ในการรับและส่งเมล์ โดยรับจากยูส - เซอร์เอเจนท์ แล้วตรวจสอบว่าผู้รับปลายทางอยู่ที่ใด หากผู้รับอยู่ภายในเครื่องเดียวกันก็จะเก็บเมล์นั้นไว้ในกล่องจดหมาย แต่หากอยู่คนละเซิร์ฟเวอร์ ก็จะต้องอาศัยโปรโตคอล SMTP ผ่านเครือข่ายทีซีพี เพื่อส่งเมล์นั้นออกไปยังเครื่องเซิร์ฟเวอร์อื่นๆ ในขณะเดียวกันก็ทำหน้าที่รับเมล์ที่ส่งเข้ามายังผู้รับในเครื่องนั้น แล้วทำการจัดส่งให้ผู้รับแต่ละคนอย่างถูกต้องด้วย โปรแกรมที่นิยมใช้กันเช่น Sendmail, Microsoft Mail เป็นต้น

2.6 ซีพียูโหลด

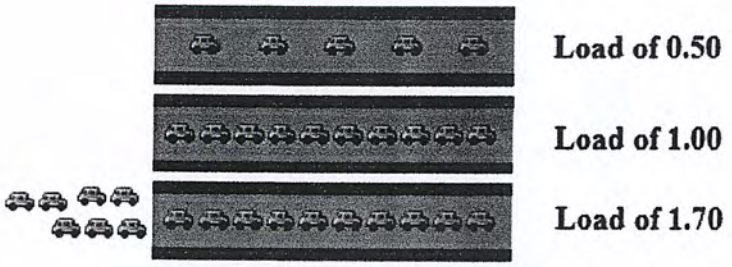
ค่าซีพียูโหลด (CPU Load) คือ ค่าเฉลี่ยของโพรเซสที่กำลังทำงานอยู่ในขณะนั้น (Load Average) โดยจะประกอบไปด้วยค่าสามค่า ซึ่งแสดงค่าเฉลี่ยของโพรเซสทั้งหมดที่ทำงานอยู่ในช่วงเวลานั้น โดยค่าเฉลี่ยตัวแรก คือค่าเฉลี่ยเมื่อหนึ่งนาทีที่แล้ว ค่าเฉลี่ยตัวที่สอง คือค่าเฉลี่ยเมื่อห้านาทีที่แล้ว และค่าเฉลี่ยตัวที่สามคือค่าเฉลี่ยเมื่อสิบห้านาทีที่แล้ว ถ้าหากค่าเฉลี่ยมีค่าต่ำ แสดงว่าระบบทำงานปกติดี แต่ถ้าค่าเฉลี่ยมีค่าสูง แสดงว่าระบบอาจมีปัญหาจากการทำงานหนัก หรือทำงานมากเกินไป

แต่การที่เราจะบอกได้ว่า ค่าเท่าไรดี หรือไม่ดีนั้น ไม่อาจอาศัยเพียงตัวเลขค่าเฉลี่ยอย่างเดียว เราจำเป็นต้องมีความเข้าใจถึงที่มาของค่าเฉลี่ยเหล่านี้ด้วย ซึ่งจะกล่าวถึงต่อไป

2.6.1 การทำงานของซีพียูกับการจราจรบนท้องถนน

ซีพียูที่มีคอร์เดียว อาจเปรียบได้กับถนนที่มีเพียงเส้นทางเดียวในการเดินรถ ถ้าหากถนนเส้นนั้นเต็มไปด้วยรถยนต์ การเคลื่อนที่ของรถคันนั้นไปยังจุดหมายก็ต้องใช้เวลาที่มากขึ้น แต่ถ้าหากถนนว่าง รถยนต์ก็สามารถวิ่งผ่านไปได้ทันที ดังนั้น ถ้าหากเราจะเปรียบถนน และรถยนต์ให้เหมือนกับการทำงานของซีพียู เราอาจเปรียบได้ดังนี้

1) ค่า 0.00 หมายความว่าบนถนนนั้นไม่มีรถยนต์แล่นอยู่เลย (ในความเป็นจริง ค่าตั้งแต่ 0.00 จนถึง 0.99 อาจกล่าวได้ว่าบนถนนเส้นนั้นไม่มีรถยนต์อยู่เลยก็ได้) เปรียบเหมือนกับซีพียูที่มีการทำงานน้อยถึงน้อยมาก
2) ค่า 1.00 หมายความว่าถนนเส้นนั้นมีรถยนต์อยู่เต็มถนนพอดี เปรียบเหมือนกับซีพียูที่มีงานต้องทำมากขึ้น ทำให้ทุกอย่างช้าลง
3) ค่ามากกว่า 1.00 หมายถึงมีรถยนต์คันอื่นๆต่อคิวยาวออกไปจากถนนเส้นนั้น ถ้าเกิดค่าขึ้นสูงถึง 2.00 อาจหมายความว่า ถนนทั้งเส้นนั้นเต็มไปด้วยรถที่กำลังวิ่งอยู่บนถนนนั้น และมีถนนอีกเส้นหนึ่งที่เต็มไปด้วยรถยนต์ซึ่งกำลังรอที่จะเข้ามายังถนนเส้นหลัก



รูป 2.6 เปรียบเทียบค่าเฉลี่ยโหลดแต่ละค่า

ดังนั้นซีพียูโหลด อาจเปรียบได้กับ รถยนต์ที่กำลังดำเนินอยู่บนถนนหลักด้วยระยะเวลาหนึ่ง หรือกำลังรอคอยที่จะเข้าสู่ถนนเส้นนั้น ในระบบปฏิบัติการยูนิกซ์ ค่านี้คือค่าของจำนวนโพรเซสทั้งหมดที่กำลังทำงานอยู่ รวมกับจำนวนของซีพียูที่กำลังรอคอยที่จะทำงาน (อยู่ในคิว) นั่นเอง

ถ้าเป็นเช่นนั้น จำนวนของรถยนต์หรือโพรเซสที่ต้องรอคอยควรจะมีค่าที่น้อยมาก หรือต่ำกว่า 1.00 ก็จะเป็นการดี (ในกรณีสำหรับซีพียู 1 คอร์) ซึ่งหมายความว่าไม่มีจำนวนของโพรเซสที่กำลังรอคิวอยู่เลย

ในการทำงานจริง เราอาจกำหนดค่าที่ต้องการให้เกิดการเตือนภัยไว้ที่ประมาณ 0.70 ในกรณีสำหรับซีพียู 1 คอร์ อันเป็นค่าที่เกิดจากการประมาณของผู้มีประสบการณ์ในเรื่องการดูแลเซิร์ฟเวอร์นั่นเอง

2.7 ระบบปฏิบัติการลีนุกซ์ (Linux Operating System)

ลีนุกซ์เป็นระบบปฏิบัติการแบบยูนิกส์ โดยใช้ลีนุกซ์เคอร์เนลในการทำหน้าที่เป็นศูนย์กลางประสานงานกับไลบรารี และเครื่องมือต่างๆ ซึ่งลีนุกซ์เป็นซอร์ฟแวร์ที่ไม่เสียค่าลิขสิทธิ์ และเป็นที่นิยมใช้ในเครื่องที่ต้องการให้บริการเป็นเซิร์ฟเวอร์ โดยสามารถทำเป็นเว็บเซิร์ฟเวอร์, เมล์เซิร์ฟเวอร์, ดาต้าเบสเซิร์ฟเวอร์, ดีเอ็นเอสเซิร์ฟเวอร์, พร็อกซีเซิร์ฟเวอร์, เอฟทีพีเซิร์ฟเวอร์ เป็นต้น ทั้งยังสามารถนำโค้ดของลีนุกซ์ไปแก้ไขได้ ซึ่งทำให้สามารถเพิ่มการทำงานบางอย่างที่เราต้องการได้ซึ่งอาจจะเป็นเรื่องไม่ง่ายสำหรับผู้เริ่มต้นแต่ก็ไม่ยากนักสำหรับผู้ที่คลุกคลีกับลีนุกซ์จนเคยชิน ลีนุกซ์มีเครื่องมือที่ใช้ดาวน์โหลดซอฟท์แวร์ อัพเดทซอฟแวร์ ได้ฟรีตลอดเวลาจากซอร์สหรือแหล่งแจกจ่ายหลักที่กำหนดไว้

ลีนุกซ์นั้นมีหลายดิสทริบิวชัน เช่น CentOS, Redhat, Gentoo, Fedora, openSUSE และ Ubuntu เป็นต้น ในที่นี้ระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส จะเน้นประยุกต์ใช้งานกับลีนุกซ์ที่เป็น Ubuntu เป็นหลัก เพื่อให้เข้าใจการทำงานของลีนุกซ์เซิร์ฟเวอร์ได้สะดวกขึ้น เนื่องจากมีเครื่องมือและผู้พัฒนาเป็นจำนวนมาก

2.8 การออกแบบระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส

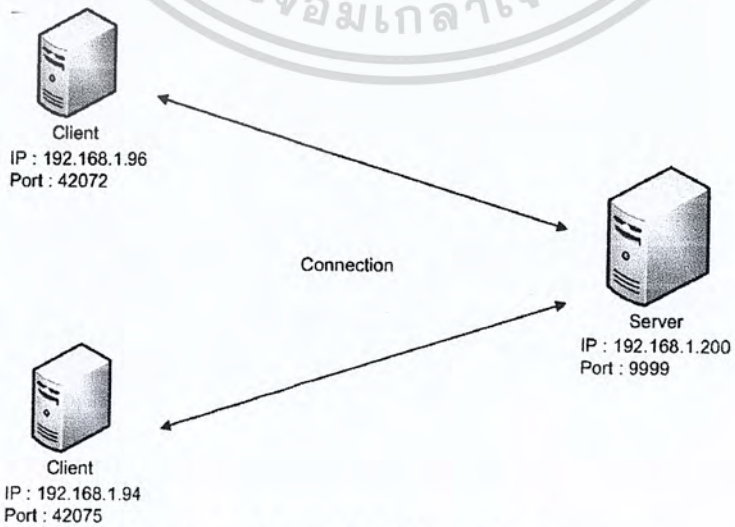
การออกแบบระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส การออกแบบระบบเฝ้าระวังจะเฝ้าระวังในส่วนของระบบเป็นหลัก อาทิ เช่น โพรเซส เซอร์วิส ซีพียู เป็นต้น โดยส่วนนี้จะถูกติดตั้งไว้ที่ไคลเอนต์ แล้วส่งข้อมูลเฝ้าระวังมายังฝั่งเซิร์ฟเวอร์ทำการประมวลผล ส่วนการเตือนภัย เป็นการแจ้งการเตือนภัยให้แก่ผู้ดูแลระบบเมื่อข้อมูลของระบบตรงตามเงื่อนไขต่างๆที่กำหนดไว้ และในส่วนการอพยพโพรเซส คือการย้ายโพรเซสจากเซิร์ฟเวอร์หนึ่ง ไปยังอีกเซิร์ฟเวอร์หนึ่งเพื่อให้การทำงานของโพรเซสสามารถดำเนินงานต่อไปได้

2.9 ภาษาที่ใช้ในการพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส

การพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส จะใช้ภาษาเพิร์ลพัฒนาเป็นหลัก เพราะเพิร์ลเป็นภาษาที่ใช้จัดการกับระบบยูนิกส์ หรือลีนุกซ์ และยังสามารถใช้คำสั่งของระบบได้อีกด้วย ไม่ว่าจะเป็นการจัดการไฟล์ด้วย grep, sed, awk เป็นต้น ดังนั้นภาษาเพิร์ล จึงเป็นที่นิยมในหมู่ผู้ทำงานด้านระบบ รวมถึงการเขียนโปรแกรมด้านระบบด้วยเช่นกัน

2.10 การติดต่อระหว่างเซิร์ฟเวอร์กับไคลเอนต์

การติดต่อระหว่างเซิร์ฟเวอร์กับไคลเอนต์ในระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส จะใช้ซ็อกเก็ตช่วยในการติดต่อสื่อสารระหว่างอุปกรณ์ผ่านเครือข่ายทีซีพีไอพี การส่งข้อมูลโดยใช้ซ็อก -เก็ตจะใช้ยูดีพีโปรโตคอลในการส่งข้อมูลจึงต้องกำหนดหมายเลขพอร์ทซึ่งเป็นเสมือนท่อระหว่างเซิร์ฟเวอร์กับไคลเอนต์เพื่อส่งข้อมูล และทั้งฝั่งเซิร์ฟเวอร์กับไคลเอนต์จะต้องสร้างซ็อกเก็ตขึ้นมาเพื่อเชื่อมต่อกันในการรับ – ส่งข้อมูล



รูป 2.7 การติดต่อระหว่างเซิร์ฟเวอร์กับไคลเอนต์

บทที่ 3
ระบบปฏิบัติการลีนุกซ์

3.1 โครงสร้างของระบบลีนุกซ์

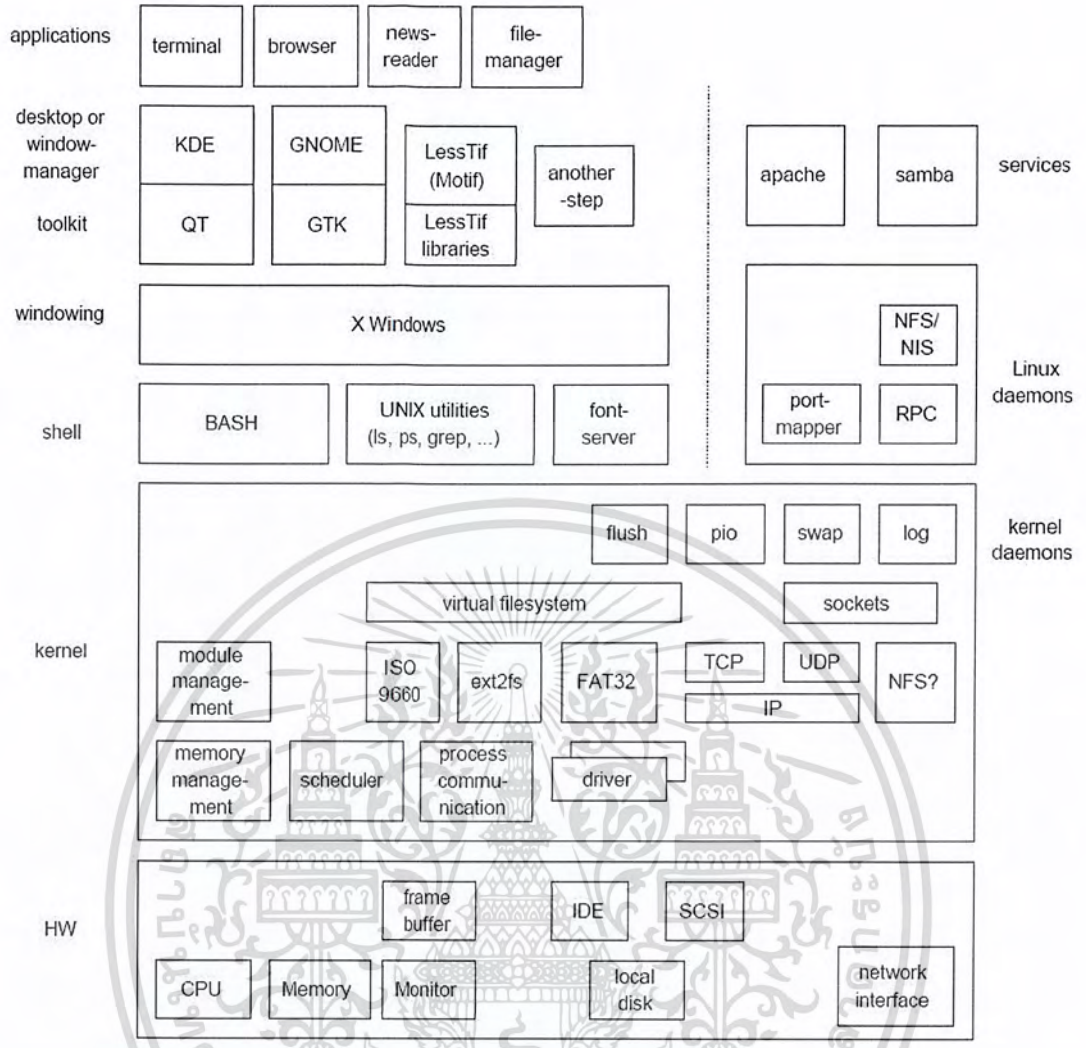
โครงสร้างพื้นฐานในการทำงานของระบบลีนุกซ์ มีด้วยกัน 4 ระดับ ดังนี้

1) แอปพลิเคชั่น (Application) หรือ Command Process คือโปรแกรมต่างๆที่เขียนขึ้นมาใช้งาน อย่างเช่น เพิร์ล, MySQL, Apache, Postfix, SendMail เป็นต้น

2) เคอร์เนล (Kernel) คือส่วนที่ใช้ควบคุมการทำงานของระบบทั้งหมด มีหน้าที่ตรวจสอบ และบริหารการใช้หน่วยความจำ ตรวจสอบ ควบคุมอุปกรณ์ฮาร์ดแวร์ โดยตัวเคอร์เนลจะขึ้นอยู่กับฮาร์ดแวร์ของแต่ละรุ่น

3) เชลล์ (Shell) คือคอมมานด์อินเตอร์พรีเตอร์ ซึ่งเป็นตัวกลางที่คอยรับคำสั่งจากผู้ใช้ผ่านทางคีย์บอร์ด แล้วแปลคำสั่งต่างๆให้เป็นภาษาเครื่อง ซึ่งจริงๆแล้วถ้าเปรียบเทียบกับระบบวินโดวส์ก็คือคอมมานด์บน DOS นั้นเอง นอกจากนี้เชลล์ยังมีความสามารถในการเขียนโปรแกรมเชลล์สคริปต์ (Shell Script) ซึ่งเป็นการนำคำสั่งต่างๆบนลีนุกซ์มาเขียนเป็นโปรแกรมเหมือนกับ batch file บน DOS

4) ฮาร์ดแวร์ (Hardware) คือตัวเครื่องคอมพิวเตอร์ ส่วนประกอบต่างๆของเครื่องคอมพิวเตอร์ อย่างเช่น ซีพียู เมนบอร์ด จอภาพ คีย์บอร์ด เมาส์ ดิสก์ ฮาร์ดดิสก์ หน่วยความจำ เป็นต้น



รูป 3.1 โครงสร้างระบบลีนุกซ์

3.2 ระบบไฟล์ของลีนุกซ์

ระบบลีนุกซ์สนับสนุนการทำมัลติยูสเซอร์ (Multi-user Operating System) โดยจำลองการทำงาน (Simultaneously) ของระบบเพื่อรองรับการเชื่อมต่อของผู้ใช้งานจำนวนมาก ระบบจะจัดการแชร์อุปกรณ์ต่างๆ เช่น เครื่องพิมพ์ เทปไดร์ฟ หน่วยความจำ และ CPU Time ให้กับผู้ใช้งานทุกคนอย่างเหมาะสม ด้วยเหตุนี้เองดีไวท์และทุกสิ่งบนระบบจึงต้องมีสิทธิ์ในการเข้าถึง (อ่าน เขียน สั่งรัน)

3.3 ประเภทของไฟล์บนลีนุกซ์

ระบบปฏิบัติการลีนุกซ์จะมีไฟล์หลายชนิดในการทำงาน มีทั้งไฟล์หลักที่ใช้ในการทำงาน และไฟล์ที่ถูกใช้โดยผู้ใช้งาน สามารถแบ่งชนิดของไฟล์ได้ดังนี้

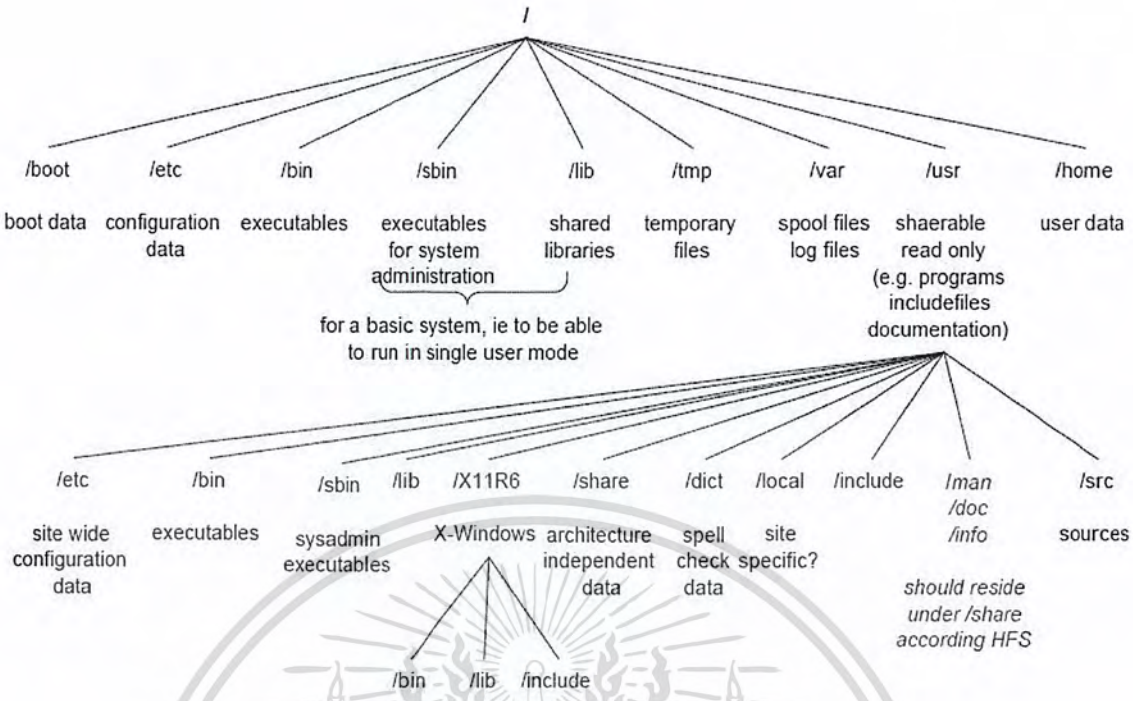
1) ไฟล์ธรรมดา (Normal File) หรือเรียกอีกชื่อหนึ่งว่า เรกูลาร์ไฟล์ (Regular File) เป็นไฟล์ที่เกิดจากการที่ยูสเซอร์สร้างขึ้น หรือโปรแกรมแอปพลิเคชั่น เช่น ไฟล์เท็กซ์ ไฟล์รูปภาพ
2) ไฟล์ไดเรกทอรี (Directory File) เป็นไฟล์ที่ถูกสร้างขึ้นสำหรับจัดกลุ่มเก็บไฟล์ข้อมูลต่างๆ ให้เป็นระเบียบ ไดเรกทอรีจะเป็นเท็กซ์ไฟล์ที่ใช้สำหรับเก็บรายชื่อของไฟล์ต่างๆ รวมทั้งค่าไอนัมเบอร์ (I-number) ของไฟล์ต่างๆที่อยู่ภายในไดเรกทอรีนี้ด้วย
3) ไฟล์พิเศษ (Special File) เป็นไฟล์ที่ระบบปฏิบัติการใช้ในการติดต่อกับอุปกรณ์ต่างๆ เช่น ดิสก์ ฮาร์ดดิสก์ เทป จอภาพ เครื่องพิมพ์ เป็นต้น โดยจะแบ่งเป็น 2 ชนิด คือ Character Device จะส่งข้อมูลทีละ 1 ตัวอักษร โดยจะใช้กับเครื่องพิมพ์ หรือ เทอร์มินัล และ Block Device จะส่งข้อมูลได้รวดเร็วเป็นบล็อก ๆ ละ 512,1024 หรือ 2048 ตัวอักษร ส่วนมากจะส่งข้อมูลให้ดิสก์ ฮาร์ดดิสก์ เทป เป็นต้น ไฟล์พิเศษพวกนี้จะเก็บในไดเรกทอรี /dev เช่น /dev/fd0 (Floppy Disk) /dev/hda1 (IDE Harddisk) /dev/sda1(SCSI Harddisk) เป็นต้น
4) ไฟล์เชื่อมโยง (Link File) เป็นไฟล์ที่มีการเชื่อมโยงกับไฟล์ต้นฉบับ ซึ่งจะเกี่ยวข้องกับไอโหนด (i-node) ของไฟล์ ลักษณะจะคล้ายๆกับการสร้างช็อตคัดให้โปรแกรม ถ้าไฟล์ต้นฉบับมีการเปลี่ยนแปลงก็จะส่งผลกระทบกับไฟล์เชื่อมโยง

```
lrwxrwxrwx  1 root     root           11 2011-02-14 12:26 mounts -> self/mounts
dr-xr-xr-x  3 root     root            0 2011-02-14 12:26 mpt
-rw-r--r--  1 root     root            0 2011-02-14 12:26 mtrr
lrwxrwxrwx  1 root     root            8 2011-02-14 12:26 net -> self/net
-r--r--r--  1 root     root            0 2011-02-14 12:26 pagetypeinfo
-r--r--r--  1 root     root            0 2011-02-14 12:26 partitions
-r--r--r--  1 root     root            0 2011-02-14 12:26 sched_debug
```

รูป 3.2 ไฟล์พิเศษที่มีการอ้างถึง

3.4 โครงสร้างไดเรกทอรีของลีนุกซ์

โครงสร้างไดเรกทอรีของระบบปฏิบัติการลีนุกซ์เป็นแบบ FHS (Filesystem Hierarchy Standard) ซึ่งมีโครงสร้างแบบต้นไม้ และมีการจัดการแบบลำดับชั้น คือ ส่วนบนสุดจะเป็นไดเรกทอรีราก (Root Directory) ใช้เครื่องหมาย "/" แทน และจะมีไดเรกทอรีย่อยที่ใช้สำหรับเก็บไฟล์ทำงานต่างๆไล่ลำดับลงมา



รูป 3.3 โครงสร้างไดเรกทอรีของลีนุกซ์

ตาราง 3.1 รายละเอียดของไดเรกเทอรี

ไดเรกทอรี	รายละเอียด
/bin	เก็บคำสั่งทั่วไป รวมทั้งยูทิลิตี้ของระบบด้วย
/boot	เก็บเคอร์เนล และไฟล์ที่ใช้บูตระบบของลีนุกซ์
/dev	เก็บไฟล์ดีไวท์สำหรับใช้อ้างอิงถึงตัวอุปกรณ์ โดยอุปกรณ์ทุกตัวจะมีไฟล์เป็นของตัวเอง อย่างเช่น /dev/tty0 คือ serial port (com1) เป็นต้น
/etc	เป็นไดเรกทอรีที่สำคัญมาก ใช้เก็บคำสั่งเพิ่มเติม และใช้เก็บไฟล์ข้อมูลประเภทคอนฟิกูเรชั่น (Configuration) และไดเรกทอรีย่อยหลักๆที่ใช้ในการเซตอัพ (Setup) ระบบ อย่างเช่น rc.local
/home	เป็นไดเรกทอรีที่ใช้เก็บไดเรกทอรี home ของยูสเซอร์ (User) บนระบบ
/lost+found	เป็นไดเรกทอรีที่ใช้เก็บ error ทั่วไปเกี่ยวกับดิสก์
/lib	เป็นไดเรกทอรีที่ใช้เก็บไลบรารี (Library) แบบ Dynamic ที่เขียนจากภาษา C มีลักษณะคล้าย .dll ของระบบปฏิบัติการวินโดวส์ (Windows)
/media	เป็นไดเรกทอรีที่ใช้เก็บไฟล์ดีไวท์สำหรับใช้อ้างอิงถึงอุปกรณ์ โดยอุปกรณ์ทุกตัวจะมีไฟล์เป็นของตัวเอง อย่างเช่น cdrom (/media/cdrom), harddisk (/media/win) เป็นต้น

ตาราง 3.1 รายละเอียดของไดเรกเทอรี (ต่อ)

ไดเรกทอรี	รายละเอียด
/proc	เป็นไดเรกทอรีสำหรับเก็บข้อมูลขณะโพรเซส (Process) ต่างๆกำลังทำงานอยู่
/sbin	เป็นไดเรกทอรีที่ใช้เก็บคำสั่งในการดูแลระบบต่างๆ
/tmp	เป็นไดเรกทอรีที่ใช้เก็บไฟล์ชั่วคราวของ root และผู้ใช้งานทั่วไป
/usr	เป็นไดเรกทอรีย่อยในการเซตอัพ (Setup) ระบบ
/usr/bin	เก็บคำสั่งทั่วไป
/usr/sbin	เก็บชุดคำสั่งเกี่ยวกับระบบเครือข่าย (Network)
/var	เป็นไดเรกทอรีที่ใช้เก็บข้อมูลที่เปลี่ยนแปลงตลอดเวลา เช่น ไดเรกทอรี www ซึ่งเป็นที่เก็บ html files หรือ log files ต่างๆ

บทที่ 4
ภาษาเพิร์ล

4.1 ความหมายของภาษาเพิร์ล

เพิร์ล ย่อมาจากคำว่า Practice Extraction and Report Language เป็นภาษาที่เขียนขึ้นเพื่อใช้งานกับระบบปฏิบัติการยูนิกส์โดยเฉพาะ ถูกสร้างขึ้นในปี 1986 โดยนาย Larry Wall ภาษาเพิร์ลเป็นภาษาที่มีรากฐานการพัฒนามาจากภาษา C และเป็นภาษาแบบสคริปต์ จึงไม่ต้องมีการคอมไพล์ซอร์สโค้ดของสคริปต์ให้เป็นโปรแกรมก่อนรัน แต่การรันสคริปต์ต้องเรียกอินเตอร์พรีเตอร์ของเพิร์ลมาอ่านสคริปต์เพื่อทำงานตามที่เขียนไว้ จริงๆแล้วการเขียนเพิร์ล นั้นควรเรียกว่า เป็นการเขียนสคริปต์มากกว่า แต่โดยทั่วไปก็นิยมเรียกว่าเป็นการเขียนโปรแกรม

4.2 ตัวแปร (Variable)

เพิร์ลมี Built in variable อยู่ 3 ชนิด คือ Scalar, Array และ Hash โดยมีรายละเอียดดังนี้

1) ถ้าเป็นตัวแปรสเกลาร์ (Scalar) ให้ใช้เครื่องหมาย $ นำหน้า
2) ถ้าเป็นตัวแปรอาร์เรย์ (Array) ให้ใช้เครื่องหมาย @ นำหน้า
3) ถ้าเป็นตัวแปรแฮช (Hash) ให้ใช้เครื่องหมาย % นำหน้า
4) ตัวที่ตามด้วยเครื่องหมาย $ และ @ ต้องเป็นตัวอักษรเท่านั้น อย่างเช่น $a, @day ไม่ใช่ $1 หรือ @+day
5) ไม่ใช้สัญลักษณ์อื่นใดรวมในตัวแปร อย่างเช่น $oho! @question?
6) ตัวอักษรตัวเล็กและตัวใหญ่ให้ความหมายเป็นคนละตัวแปร

4.2.1 สเกลาร์ (Scalar)

เป็นตัวแปรที่ใช้เก็บข้อมูลเพียงข้อมูลเดียว ซึ่งเป็นตัวเลข (Integer) หรือข้อความ (String) ก็ได้

4.2.1.1 ตัวแปรสตริง

Single-Quote ‘’ ถ้าใช้เครื่องหมายนี้ล้อมรอบข้อความ หมายความว่า ตัวอักษรทุกตัวที่อยู่ข้างในเป็นเนื้อหาข้อความจริงๆ โดยอินเตอร์พรีเตอร์จะไม่สนใจที่จะตีความหมายของตัวอักษรพิเศษที่แทรกอยู่ แต่ถ้าหากต้องการใช้ตัวอักษรพิเศษ เป็นส่วนหนึ่งของข้อความ อย่างเช่น don’t talk จะต้องใส่เครื่องหมาย \ กำกับหน้าตำแหน่งของเครื่องหมาย Single-Quote ในข้อความนั้นด้วย

ตัวอย่าง 4.1 การใช้ Single quote กับตัวแปรสตริง

```
$name1 = 'harry\n';     ผลลัพธ์คือ harry\n
$string1 = 'Don't talk';ผลลัพธ์จะผิดและแสดงข้อความเป็น error
$string2 = 'Don\'t talk';      ผลลัพธ์คือ Don't talk
```

Double-Quote " " ถ้าใช้เครื่องหมายนี้ล้อมรอบข้อความ จะหมายความว่า ตัวอักษรทุกตัวที่อยู่ข้างในข้อความมีตัวอักษรพิเศษแทรกอยู่ ถ้าอินเตอร์พรีเตอร์พบเครื่องหมาย " " จะนำเข้าสู่การประมวลผล อย่างเช่น การค้นหาคำที่แยกเป็นตัวอักษร ฉะนั้นถ้าในข้อความไม่มีตัวอักษรพิเศษแทรกอยู่ ก็ไม่ควรใช้เครื่องหมาย " " เพื่อการประหยัดทรัพยากรของเครื่องเพื่อที่จะได้ประมวลผลเร็วขึ้น

ตัวอย่าง 4.2 การใช้ Double quote กับตัวแปรสตริง

```
$name2 = "harry\n";    ผลลัพธ์ harry พร้อมขึ้นบรรทัดใหม่
```

Back-Quote ` ` ถ้าใช้เครื่องหมายนี้ล้อมรอบข้อความ นั่นหมายความว่า ตัวอักษรทุกตัวที่อยู่ข้างในข้อความสามารถรันได้ ซึ่งเป็นคำสั่งในยูนิกส์ อย่างเช่น คำสั่ง date ในยูนิกซ์คือการแสดงเวลาของเครื่องคอมพิวเตอร์

ตัวอย่าง 4.3 การใช้ Back quote เพื่อเรียกคำสั่งในระบบปฏิบัติการยูนิกซ์

```
$datetime = `date`'      ผลลัพธ์คือ Mon Feb 14 03:31:16 ICT 2011
```

4.2.1.2 ตัวแปรตัวเลข

ข้อมูลที่เป็นตัวเลข ไม่นิยมใช้เครื่องหมาย " " หรือ ' ' นั่นคือจะไม่ใส่เครื่องหมายใดๆ อย่างเช่น $phone = 7;

4.2.2 อาร์เรย์ (Array)

เป็นตัวแปรที่ใช้เก็บชุดข้อมูล ซึ่งเป็นการเก็บข้อมูลแบบเป็นชุด เป็นแถว หรือกลุ่มของข้อมูล อย่างเช่น วันจะประกอบไปด้วย จันทร์ – อาทิตย์ การเก็บข้อมูลในรูปแบบนี้ เพิร์ลจะใช้เครื่องหมาย @ นำหน้า ซึ่งการกำหนดค่าให้กับตัวแปรอาร์เรย์ (Array) เราสามารถทำได้โดยการเพิ่ม หรือลดค่าตัวแปรได้ ดังนี้

ตัวอย่าง 4.4 การเพิ่มค่าในอาร์เรย์

```
@day = ('Mon','Tue');
@add = ('Wed','Thur','Fri','Sat','Sun');
@day = ('Mon','Tue',@add);
```

หรืออาจใช้การเพิ่มค่าอาร์เรย์ด้วยฟังก์ชัน push หรือ pop ก็ได้

ตัวอย่าง 4.5 การเพิ่มค่าในอาร์เรย์ด้วยฟังก์ชั่น push, pop

```
push (ข้อมูลตัวแปรเก่า, ข้อมูลตัวแปรใหม่)
pop (ข้อมูลตัวแปรเก่า)
```

4.2.3 แฮช (Hash)

เป็นตัวแปรที่ใช้เก็บข้อมูลแบบเรียงลำดับ ซึ่งสมาชิกแต่ละตัวจะมีค่า key และค่า value โดยค่า key เปรียบเสมือนค่า index วิธีการกำหนดตัวแปรแบบนี้ เราจะใช้เครื่องหมาย % นำหน้าเสมอ

ตัวอย่าง 4.6 การอ้างถึงค่า key ของ Mon และป้อนค่า key ของ Mon ตามลำดับ

```
$week{'Mon'}
%week = (Mon=>1,Tue=>2);
```

4.3 เครื่องหมายดำเนินการ (Operator)

เครื่องหมายดำเนินการ Operator ของเพิร์ล มีด้วยกัน 6 แบบ ดังนี้

4.3.1 เครื่องหมายคณิตศาสตร์ (Arithmetic Operators)

ตาราง 4.1 เครื่องหมายคณิตศาสตร์

ชื่อ	Operator
บวก	+
ลบ	-

ตาราง 4.1 เครื่องหมายคณิตศาสตร์ (ต่อ)

ชื่อ	Operator
คูณ	*
หาร	/
เศษที่เกิดจากการหาร	%
ยกกำลัง	**

4.3.2 เครื่องหมายทางการกำหนดค่า (Assignment Operators)

ตาราง 4.2 เครื่องหมายการกำหนดค่า

Operator	ตัวอย่าง	ความหมาย
=	$tmp = 5	ให้ตัวแปร tmp มีค่าเป็น 5
++	$tmp ++ หรือ ++$tmp	เพิ่มค่าของตัวแปร tmp ขึ้นอีก 1 และเก็บใน tmp
--	$tmp -- หรือ --$tmp	ลดค่าของตัวแปร tmp ลงอีก 1 และเก็บใน tmp
+=	$tmp += 3	เพิ่มค่าของตัวแปร tmp ขึ้นอีก 3 และเก็บใน tmp
-=	$tmp -= 2	ลดค่าของตัวแปร tmp ลงอีก 2 และเก็บใน tmp
.= (. คือ concatenate)	$tmp .= “ing”	นำข้อความ “ing” เพิ่มต่อท้ายเข้าไปในตัวแปร tmp
*=	$tmp *= 4	คูณค่าในตัวแปร tmp ด้วย 4 และเก็บในตัวแปร tmp
/=	$tmp /= 2	นำค่าในตัวแปร tmp หารด้วย 2 และเก็บในตัวแปร tmp
**=	$tmp **= 2	นำค่าตัวแปร tmp ยกกำลัง 2 และเก็บในตัวแปร tmp
x= (x คือซ้ำค่า)	$tmp x= 20	ซ้ำค่าในตัวแปร $tmp 20 รอบ และเก็บในตัวแปร tmp

4.3.3 เครื่องหมายดำเนินการกับข้อความ (String Operators)

ตาราง 4.3 เครื่องหมายดำเนินการกับข้อความ

Operator	คำอธิบาย
.	ต่อสตริง
x	ทำซ้ำสตริง
substr()	substr($str1,$n1,$n2) คือตัดข้อความใน $str1 ที่ตำแหน่ง $n1 เป็นความยาว $n2
index()	Index($n1,$n2) คือ ค้นหาว่าตัวแปร $n2 อยู่ตำแหน่งไหนของตัวแปร $n1 ถ้ามีจะบ่งตำแหน่งมาเป็นตัวเลข ถ้าไม่มีผลลัพธ์เป็น -1

4.3.4 เครื่องหมายดำเนินการเปรียบเทียบ (Relational Operators)

ตาราง 4.4 เครื่องหมายดำเนินการเปรียบเทียบ

กับค่าตัวเลข	กับข้อความ	ความหมาย
==	eq	เท่ากันกับ
!=	ne	ไม่เท่ากับ
>	gt	มากกว่า
>=	ge	มากกว่าหรือเท่ากับ
<	lt	น้อยกว่า
<=	le	น้อยกว่าหรือเท่ากันกับ
<=>	cmp	เปรียบเทียบสองข้าง

4.3.5 เครื่องหมายดำเนินการค้นหา/แทนที่ (Pattern Matching Operators)

ตาราง 4.5 เครื่องหมายดำเนินการค้นหา หรือแทนที่คำเหมือน

Operator	ตัวอย่าง	ความหมาย
=~//	$tmp =~ /harry/	ให้ค่าเป็นจริงถ้าหาก $tmp มี “harry”
=~ s///	$tmp =~ s/harry/peter/	เปลี่ยน ‘harry’ ใน $tmp เป็น ‘peter’
=~ tr///	$tmp =~ tr/a-z/A-Z/	แปลงตัวอักษรตามที่กำหนด

4.3.6 เครื่องหมายดำเนินการทางตรรกะ (Logical Operators)

ตาราง 4.6 เครื่องหมายดำเนินการทางตรรกะ

Operator	ตัวอย่าง	ความหมาย
&&	$a && $b	ให้ผลเป็นจริงเมื่อ $a และ $b เป็นจริงทั้งคู่ (and)
||	$a || $b	ให้ผลเป็นจริงเมื่อตัวใดตัวหนึ่งเป็นจริง
!	! $a	ให้ผลเป็นจริงเมื่อ $a เป็นเท็จ

4.4 การทำงานที่เป็นเงื่อนไข และการควบคุมการวนรอบ (Loop Control)

การทำงานที่เป็นเงื่อนไขและลูปของภาษาเพิร์ลไม่มีความแตกต่างจากภาษาอื่นมากนัก อาจมีเพียงบางคำสั่งที่มีเพิ่มเติมออกมา แต่ทั้งนี้ สามารถศึกษาได้จากภาษาอื่นๆเช่นกัน

4.5 โปรแกรมย่อย (Sub Program)

โดยทั่วไปโปรแกรมย่อยจะมี 2 แบบคือ Procedure และ Function แต่ในเพิร์ล (Perl) มีเพียงแบบเดียวคือ Sub Program คำขึ้นต้นของโปรแกรมย่อยคือ “sub” เพื่อเป็นการประกาศว่า เป็นโปรแกรมย่อย และเขียนตามด้วยชื่อของโปรแกรมย่อยที่ตั้งขึ้น ซึ่งชื่อของโปรแกรมย่อยนี้ห้ามตั้งซ้ำกับชื่อของคำสั่ง หรือฟังก์ชั่นของเพิร์ลที่มีอยู่แล้ว ตัวแปรภายใน (Local Variable) มี 2 แบบ คือ ใช้คำว่า my หรือ local นำหน้าชื่อตัวแปร ความแตกต่างกันคือ ถ้าประกาศด้วยคำว่า my จะมองเห็นและใช้งานได้เฉพาะภายในโปรแกรมย่อยที่ประกาศเท่านั้น โปรแกรมย่อยอื่นจะมองไม่เห็น แต่ถ้าหากว่าใช้คำว่า local นำหน้าตัวแปร โปรแกรมย่อยทั้งหมดจะมองเห็นและใช้งานได้ด้วย แต่ถ้าเราไม่ใช้คำว่า my หรือ local นำหน้าตัวแปร ตัวแปรเหล่านี้จะกลายเป็นแบบ global ไปโดยอัตโนมัติ

โปรแกรมย่อย (Sub Program) ในเพิร์ล สามารถเขียนไว้ในไฟล์เดียวกันกับโปรแกรมหลัก ที่เรียกโปรแกรมย่อยให้ทำงาน หรือจะเขียนโปรแกรมย่อยแยกไว้ในไฟล์อื่นก็ได้ นอกจากนี้โปรแกรมย่อยในเพิร์ลสามารถเขียนไว้ในส่วนไหนของโปรแกรมหลักก็ได้ แต่ส่วนใหญ่นิยมเขียนโปรแกรมย่อยไว้ส่วนบน หรือส่วนล่างของโปรแกรมหลัก

4.5.1 การเรียกโปรแกรมย่อย

ให้ระบุเครื่องหมาย & ไว้หน้าชื่อของโปรแกรมย่อย

โปรแกรม 4.1 การเรียกใช้โปรแกรมย่อย

```
sub get_input{
   do{
      $input = <STDIN>;
      Chomp($input);
   }
   While($input<0);
}
#main prog
$sum = 0;
for($i=1;$i<=2;$i++){
   get_input;
   $sum += $input;
}
print "sum = $sum";
```

สำหรับเครื่องหมาย & จะใส่หรือไม่ใส่ก็ได้ ขึ้นอยู่กับว่าโปรแกรมย่อยนั้นวางอยู่ตรงไหนของโปรแกรมหลัก ถ้าหากว่าโปรแกรมย่อยนั้นวางไว้ในส่วนบนของโปรแกรม เมื่ออินเตอร์พรีเตอร์ทำงานผ่านโปรแกรมย่อยก่อนจนมาเจอการเรียกใช้โปรแกรมย่อยในตัวโปรแกรมหลัก อินเตอร์พรีเตอร์ก็จะเข้าใจว่าเป็นชื่อของโปรแกรมย่อย และทำการเรียกใช้โปรแกรมย่อยได้อย่างถูกต้อง โดยไม่ต้องระบุเครื่องหมาย & ไว้ที่ส่วนหน้าของชื่อโปรแกรมย่อยก็ได้ แต่ว่าถ้าวางโปรแกรมย่อยไว้ส่วนล่างของโปรแกรมหลัก และอินเตอร์พรีเตอร์ทำงานมาจนเจอชื่อของโปรแกรมย่อย อินเตอร์พรีเตอร์จะไม่ทราบว่าเป็นชื่อของโปรแกรมย่อย จำเป็นจะต้องระบุเครื่องหมาย & ไว้ที่หน้าชื่อของโปรแกรมย่อย เพื่อให้อินเตอร์พรีเตอร์รู้ว่าเป็นชื่อของโปรแกรมย่อย

4.5.2 การส่งค่าเข้าไปในโปรแกรมย่อย

การส่งผ่านค่าจากโปรแกรมหลักเข้ามาภายในโปรแกรมย่อยได้นั้น ในโปรแกรมย่อยจะต้องมีตัวแปรภายในมารับเท่ากับจำนวนตัวแปรที่ส่งเข้ามา เริ่มต้นตัวแปรที่ถูกส่งเข้ามาจะถูกเก็บไว้ในตัวแปร @_ ซึ่งเป็นตัวแปรสาธารณะโดยอัตโนมัติ นั่นคือ ตัวแปรพิเศษ @_ เป็นตัวแปรแบบอาร์เรย์สำหรับเก็บข้อมูลทุกตัวที่ถูกส่งมาตามลำดับเก็บไว้ในตำแหน่งของอาร์เรย์ตามลำดับเช่นกัน จากนั้นในโปรแกรมย่อยต้องประกาศตัวแปรภายในเพื่อรับค่าต่อจากตัวแปร @_ อีกทีหนึ่ง

สำหรับการส่งค่าออกจากโปรแกรมย่อยเราสามารถใช้คำสั่ง return เหมือนกับการเขียนโปรแกรมภาษาอื่นๆ หรือจะส่งที่ท้ายบรรทัดของโปรแกรมย่อยก็ได้

4.5.3 การรวมโปรแกรมย่อยไว้ด้วยกัน และใช้ require เพื่อเรียกใช้

เราสามารถรวมโปรแกรมย่อยเข้าไว้ด้วยกันในไฟล์ต่างหาก ซึ่งจะเรียกว่าเป็นไฟล์โปรแกรมย่อย แยกออกจากไฟล์ที่เป็นโปรแกรมหลัก ในไฟล์ซึ่งเก็บโปรแกรมย่อยเหล่านี้ บรรทัด

ล่างท้ายสุดในไฟล์โปรแกรมย่อยจะต้องส่งค่ากลับเป็นค่าอะไรก็ได้ที่ไม่ใช่ 0 (มีค่าเป็นจริงเท่านั้น) ส่วนมากนิยมเขียน 1 เช่น เขียน return 1 ไว้ที่ท้ายบรรทัด เป็นต้น ส่วนในโปรแกรมหลัก จะใช้คำสั่ง require ตามด้วยชื่อไฟล์ที่บรรจุโปรแกรมย่อยไว้ เหมือนกับคำสั่ง include ในภาษา C ซึ่งคือการไปดึงเอาโปรแกรมย่อยที่อยู่ในไฟล์อื่นมาใช้ เสมือนว่าอยู่ไฟล์เดียวกันนั่นเอง

4.6 การจัดการไฟล์ในเพิร์ล

4.6.1 การเปิด ปิดไฟล์ เพื่ออ่าน และเขียนทับ

เราจะใช่คำสั่ง open สำหรับการเปิดไฟล์ขึ้นมาอ่าน และ close สำหรับการปิดไฟล์ ในกรณีที่จะเขียนลงไฟล์ มักจะใช้คำสั่ง print ในระหว่างของการเปิดไฟล์ เพื่อให้ข้อมูลที่แสดงออกมาไปทับค่าที่อยู่ในไฟล์นั้น

โปรแกรม 4.2 การเปิด ปิด อ่าน และเขียนไฟล์

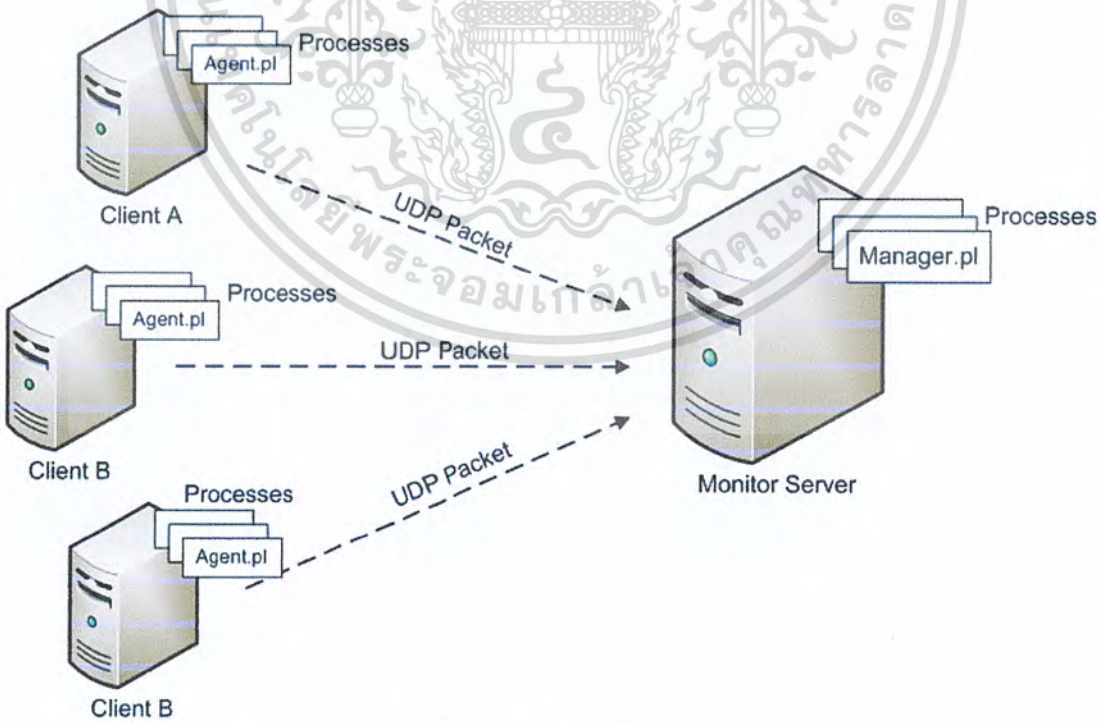
```
$data = 20;
open(File,">data.txt");  #open a file call data.txt
   print File $data;     #write $data into file
   @readfile = <File>;   #keep reading file
close(File)              #closed file
print "@readfile";
```

ในกรณีที่เราต้องเขียนต่อจากไฟล์เก่า จะใช้เครื่องหมาย ">>" แทน ">" และในบางครั้งหากเปิดไฟล์ไม่ได้ อาจใช้คำสั่ง die ช่วยในการตรวจสอบไฟล์ที่กำลังจะเปิดนั้น เพื่อให้แสดงค่าความผิดพลาดออกมาทางหน้าจอก็ได้

บทที่ 5
การออกแบบระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส

โมเดลของระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสเป็นไคลเอนต์เซิร์ฟเวอร์โมเดล ประกอบด้วยเครื่องเซิร์ฟเวอร์ที่ต้องการเฝ้าระวังในฝั่งไคลเอนต์ ซึ่งมีข้อมูลของระบบที่เราต้องการ เช่น พื้นที่เหลือในดิสก์ และการใช้หน่วยความจำ เป็นต้น เพื่อนำข้อมูลดังกล่าวมาวิเคราะห์ประสิทธิภาพการทำงาน และปัญหาที่เกิดขึ้น และมีเครื่องเซิร์ฟเวอร์ในฝั่งเซิร์ฟเวอร์ที่ทำหน้าที่ประมวลผล แสดงผล ข้อมูลเหล่านั้น ซึ่งเซิร์ฟเวอร์ตัวนี้เรากำหนดให้มีความเสถียรของระบบภายใน และทนทานต่อผลกระทบจากภายนอก เช่น ไฟตก ไฟดับ กำหนดให้มีแหล่งจ่ายไฟสำรอง เป็นต้น และโมเดลระบบมอนิเตอร์นี้เหมาะสำหรับจำนวนเครื่องเซิร์ฟเวอร์ในระดับองค์กรขนาดเล็ก

ภายในเครื่องเซิร์ฟเวอร์เหล่านั้นจะประกอบด้วยโปรแกรม หรือโพรเซสที่ทำงานตามจุดประสงค์ของระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส ซึ่งโพรเซสเหล่านี้ทำอย่างไม่ซับซ้อน เพื่อไม่เป็นภาระงานเพิ่มขึ้นให้กับเซิร์ฟเวอร์



รูป 5.1 มอนิเตอร์โมเดล

5.1 เอเจนท์

เอเจนท์ของระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส จะถูกติดตั้งไว้ที่ฝั่งไคลเอนต์ โดยเอเจนท์จะเป็นโพรเซสหนึ่งที่ทำการตรวจสอบ เฝ้าดูระบบตามรายการที่กำหนดไว้ในคอนฟิคกูเร - ชันไฟล์ แล้วส่งข้อมูลจากการมอนิเตอร์กลับไปยังฝั่งเซิร์ฟเวอร์ เพื่อให้ฝั่งเซิร์ฟเวอร์ทำการประมวลผลและแสดงผล นอกจากนี้เอเจนท์ยังมีความสามารถในการตัดสินใจแก้ปัญหาอย่างง่ายบางอย่างด้วยโพรเซสเอเจนท์เอง เช่น หากมีโพรเซสในเครื่องนั้นไม่ตอบสนอง เอเจนท์จะทำการฆ่าโพรเซสและ สั่งเริ่มการทำงานของโพรเซสนั้นใหม่ ทั้งนี้ขึ้นอยู่กับความสำคัญของโพรเซสนั้นๆ ด้วย

การทำงานร่วมกับเมเนเจอร์ในกรณีที่ต้องทำการอพยพโพรเซส ซึ่งในระดับที่เราสนใจคือการอพยพโพรเซสในรูปแบบของการทำการกระจายโหลดไปให้เซิร์ฟเวอร์อื่น สิ่งที่เอเจนท์ต้องกระทำคือการแจ้งกับเมเนเจอร์ในเรื่องของโหลด ซึ่งเป็นหน้าที่ปกติของเอเจนท์ที่จะส่งข้อมูลไปให้เมเนเจอร์อยู่แล้ว และแจ้งเตือนภัยว่าทรัพยากรของเซิร์ฟเวอร์ที่เอเจนท์อยู่นั้นมีปัญหา ซึ่งในกรณีนี้เป็นปัญหาที่เอเจนท์เองไม่สามารถแก้ปัญหาดังกล่าวได้ด้วยตัวเอเจนท์เอง ต้องอาศัยเมเนเจอร์ซึ่งเป็นผู้จัดการคำร้องที่เข้ามาจากยูเซอร์ และส่งต่อไปให้เซิร์ฟเวอร์ที่ถูกกำหนดมาให้รองรับคำร้องเหล่านั้นตามอัลกอริทึมที่กำหนดไว้ในเมเนเจอร์ และอัลกอริทึมที่ใช้สามารถเปลี่ยนแปลงได้ตามสถานการณ์หรือภาระที่เกิดขึ้นกับเซิร์ฟเวอร์ที่ให้บริการ

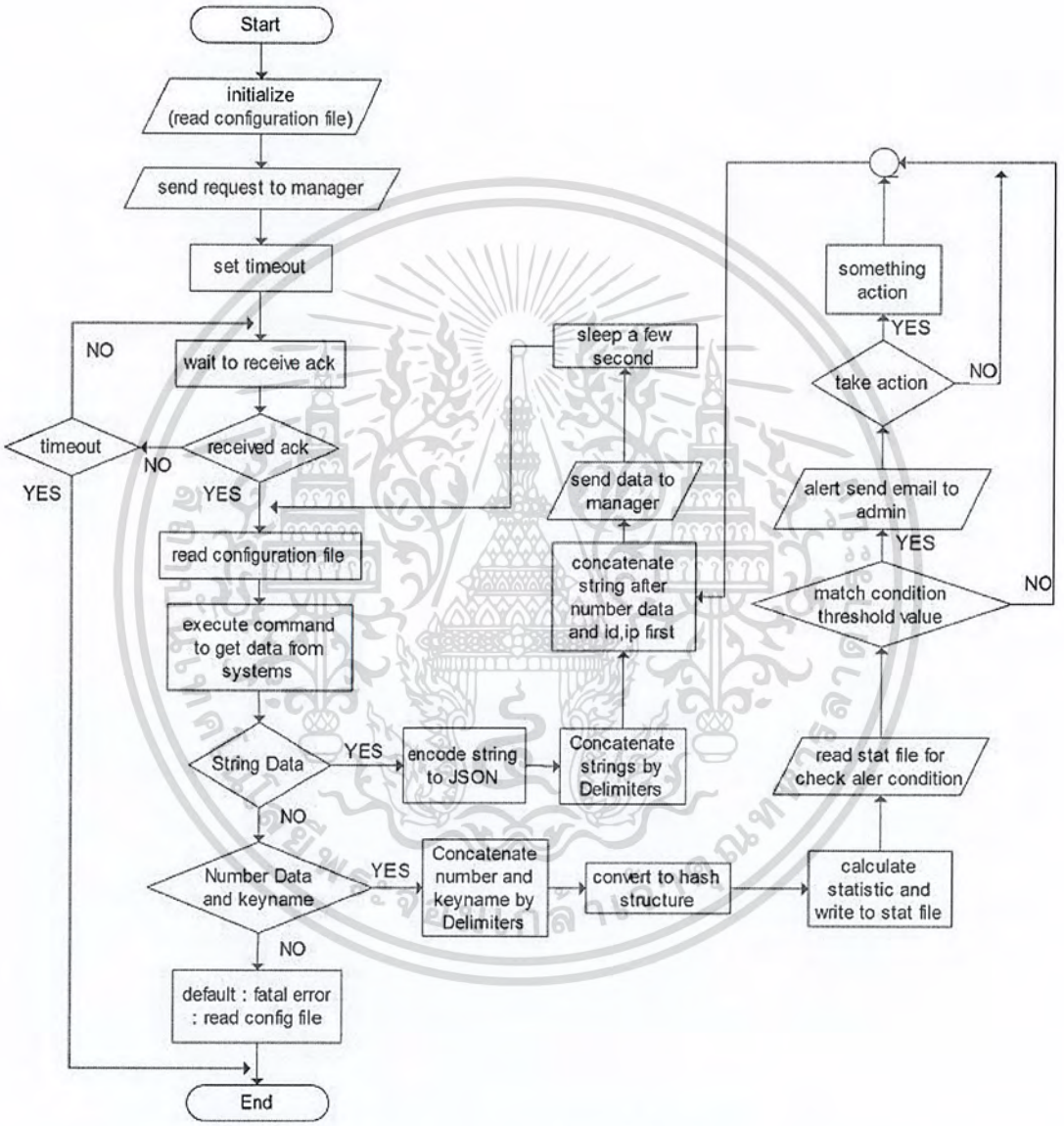
ในกรณีที่โพรเซสเอเจนท์เกิดปัญหา ระบบจะสามารถทราบได้จากโพรเซสที่มันกำลังสื่อสารอยู่ด้วยซึ่งก็คือเมเนเจอร์โพรเซสที่อยู่ฝั่งเซิร์ฟเวอร์ตามเงื่อนไขหรือข้อตกลงที่กำหนดไว้ เพื่อเตือนภัยให้ผู้ดูแลระบบทราบ และดำเนินการแก้ไขปัญหาดังกล่าวในเวลาต่อมา

5.2 เมเนเจอร์

เมเนเจอร์ของระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส จะนำข้อมูลที่ได้รับจากฝั่งเอเจนท์มาทำการประมวลผล และแสดงรายงานแก่ผู้ดูแลระบบ การประมวลผลที่ฝั่งเซิร์ฟเวอร์ค่อนข้างที่จะมีความซับซ้อนเนื่องจากรับข้อมูลมาจากเอเจนท์หลายตัว เมเนเจอร์จะต้องมีความสามารถแยกแยะหรือคัดกรองข้อมูลเพื่อเก็บลงฐานข้อมูล และสามารถทำการดึงข้อมูลขึ้นมาแสดงผลได้อย่างถูกต้อง เมเนเจอร์จะแบ่งการเก็บข้อมูลเป็น 2 ลักษณะหรือ 2 ประเภท คือ ข้อมูลที่เป็นข้อมูลเชิงปริมาณโดยตรง เช่น เปอร์เซ็นต์การใช้ซีพียู ขนาดของฮาร์ดดิสก์ที่เหลืออยู่ เป็นต้น และข้อมูลที่เป็นสตริงข้อมูล เช่น รายการของโพรเซสที่รันอยู่ รายละเอียดของซีพียูที่ใช้ เป็นต้น โดยในส่วนของการเก็บข้อมูลลงฐานข้อมูลนั้น เพื่อนำข้อมูลมาแสดงผลตามเงื่อนไข และสิทธิในการแสดงข้อมูลเครื่องเซิร์ฟเวอร์ที่กำลังมอนิเตอร์อยู่ ซึ่งเว็บเซิร์ฟเวอร์จะเป็นเครื่องเดียวกับที่เมเนเจอร์โพรเซสรันอยู่เสมอเหตุผลเพราะหน้าที่อย่างหนึ่งของเมเนเจอร์คือแสดงผลข้อมูลที่ได้รับมาซึ่งใช้เว็บในการ

แสดงผลเหล่านั้น ทั้งนี้เว็บเซิร์ฟเวอร์กับเมเนเจอร์โพรเซสสามารถอยู่บนเครื่องคนละเครื่องที่มีการเชื่อมต่อกันผ่านเน็ตเวิร์คได้

5.3 ขั้นตอนการทำงานของเอเจนท์



รูป 5.2 การทำงานของเอเจนท์โพรเซส

ขั้นตอนการทำงานของเอเจนท์นั้นเริ่มจากการอ่านคอนฟิคกูเรชันไฟล์ซึ่งในขณะติดตั้งเอ - เจนท์นั้นเราจำเป็นต้องคอมฟิกไฟล์ดังกล่าวเพื่อให้เอเจนท์ปฏิบัติตามสิ่งที่เรากำหนดได้ จากนั้นเอ - เจนท์จะทำการส่งแพ็คเก็ตคำร้อง (ซึ่งเป็นตัวเดียวกันกับฮัลโหลแพ็คเก็ตที่ใช้เช็คว่าเมเนเจอร์ยัง

สามารถตอบสนองได้อยู่หรือไม่) แต่ก่อนจะส่งแพ็คเก็ตดังกล่าวนั้นจะต้องมีการตั้งค่าไทม์เอาท์ (Time out) เพื่อไม่ให้โพรเซสนั้นเปิดการรอรับค้างไว้ตลอดเวลา ในกรณีที่ไม่ได้รับการตอบกลับภายในเวลาที่เรากำหนดไว้จะถือว่าขณะนั้นไม่สามารถติดต่อกับเมเนเจอร์ได้ในเวลาดังกล่าว ให้เอเจนท์ปิดการทำงานของตัวเองลง และหันไปตรวจสอบที่ฝั่งเมเนเจอร์หรือความหนาแน่นในเครือข่ายแทน

แต่เมื่อมีการตอบสนองจากเมเนเจอร์นั่นหมายถึงเอเจนท์ และเมเนเจอร์โพรเซสพร้อมที่จะทำการส่งข้อมูลหากันได้แล้ว เอเจนท์จะทำการอ่านคอนฟิคกูเรชันเกี่ยวกับข้อมูลที่มันจะต้องส่งไปให้เมเนเจอร์ตามการคอนฟิคของผู้ใช้งานโปรแกรม เมื่อทำการอ่านข้อมูลมาแล้ว ข้อมูลที่อ่านได้ส่วนหนึ่งจะประกอบด้วยคอมมานด์ของลีนุกซ์ที่จะทำงาน (Execute) เพื่อให้ได้ข้อมูลตามที่ต้องการ

เมื่อได้ข้อมูลมาจากระบบแล้วเอเจนท์จะทำการแยกประเภทข้อมูลออกเป็นข้อมูลที่เป็นตัวเลขและข้อมูลที่เป็นสตริงข้อมูล ในส่วนของข้อมูลที่เป็นตัวเลข ก่อนที่เราจะส่งไปให้เมเนเจอร์เราต้องทำการเก็บสถิติก่อน เพื่อวิเคราะห์ว่าสถานะของค่าแต่ละค่าตอนนี้เป็นเช่นไรนับตั้งแต่โพรเซสรันขึ้นมา และทำการแจ้งเตือนหรือเตือนภัย เมื่อมีค่าที่ตรงกับเงื่อนไขการเตือนภัยตามค่าเกณฑ์ที่เราตั้งขึ้นมาผ่านคอนฟิคกูเรชันไฟล์นั่นเอง เมื่อเก็บสถิติแล้ว ก็จะมีการอ่านสถิตินั้นขึ้นมาตรวจสอบทุกครั้งที่มีการดึงข้อมูลจากระบบ และทำการเตือนภัยดังที่กล่าวข้างต้น

การเตือนภัยของเอเจนท์นั้นคือการส่งอีเมล์ไปหาผู้ดูแลระบบ เพื่อรายงานว่ามีค่าบางอย่างตรงตามเงื่อนไขที่ระบุไว้ว่าถ้าเกิดเหตุการณ์เช่นนี้ให้แจ้งให้ผู้ดูแลระบบทราบด้วย ในขณะเดียวกันหากเป็นการเตือนภัยขั้นร้ายแรงที่ต้องการการแก้ไขก่อนที่จะสายเกินแก้ หรือแก้ไขให้อัตโนมัติโดยที่ผู้ดูแลไม่จำเป็นต้องรีโมทเข้ามาแก้ไขด้วยตนเอง

ในการส่งข้อมูลไปหาเมเนเจอร์นั้นจะมีรูปแบบของข้อมูลที่ชัดเจนอย่างง่าย คือนำไอดี ไอพี ข้อมูลตัวเลข ข้อมูลสตริง ตามลำดับ มาต่อกันคั่นด้วยสัญลักษณ์ที่ทำให้ทั้งสองฝั่งนั้นเข้าใจตรงกันว่าส่วนใดเป็นข้อมูลอะไร ซึ่งจะแสดงรูปแบบอย่างง่าย ดังนี้

ตัวอย่าง 5.1 รูปแบบของการส่งข้อมูล

```
id:-:agent ip:-:number data:-:;string data
```

เมื่อทำการส่งข้อมูลไปแล้ว เนื่องจากเป็นการส่งแบบยูดีพี ดังนั้นเราไม่สามารถบอกได้ว่าข้อมูลส่งถึงหรือไม่ และจะทำการส่งข้อมูลชุดถัดไปในทันที หรือตามระยะเวลาที่กำหนดในการส่งแต่ละรอบตามคอนฟิคกูเรชัน อย่างไรก็ตามแม้จะเป็นการส่งแบบยูดีพี เราก็สมควรที่จะรู้ว่าเมเนเจอร์นั้นยังมีตัวตนอยู่หรือไม่ หรือยังสามารถติดต่อได้ด้วยหรือไม่ ผ่านทางฮัลโหลแพ็คเก็ตที่ซ่อนอยู่ในการส่งแต่ละครั้ง

5.3.1 คอนฟิคกูเรชันไฟล์

```
#161.246.34.191:-:161.246.34.195:-:sID.123456789012196:-:9770
!interval::1
!teshold::80
#teshold::sCPU
$userCPU::sar -u 1 1 | sed -n 4,4p | awk '{print $4}'
$sysCPU::sar -u 1 1 | sed -n 4,4p | awk '{print $6}'
$load1::w | sed -n 1,1p | cut -d, -f4 | awk '{print $3}'
$load5::w | sed -n 1,1p | cut -d, -f5 | sed s/' '//
$load15::w | sed -n 1,1p | cut -d, -f6 | sed s/' '//
$pUsedHdd::df | grep '\/$' | awk '{if($1~/sda/)$4=$5};END{print $4}' | sed s/%//
$usedHdd::df | grep '\/$' | awk '{if($1~/sda/)$2=$3};END{print $2}' | sed s/%//
$freeHdd::df | grep '\/$' | awk '{if($1~/sda/)$3=$4};END{print $3}' | sed s/%//
$usedMem::cat /proc/meminfo | grep 'MemTotal' | awk -v free=$(cat /proc/meminfo | grep 'MemFree'
$freeMem::cat /proc/meminfo | grep 'MemFree' | awk '{print $2}'
$apacheload::apache2ctl status | grep 'CPU load' | awk {'print $8'} | cut -d % -f1
$apacheReq::apache2ctl status | grep 'requests/sec' | awk '{print $1}'
$totalAccess::apache2ctl status | grep 'Traffic' | awk '{print $3}'
$totalTraffic::apache2ctl status | grep 'Traffic' | awk '{print $7}'
#apacheUPTIME::apache2ctl status | grep 'uptime' | awk {'print $3"."$5"."$7"."$9'}
#totalapachePS::top -b -n 1 | grep -vE 'top|PID|Cpu|Mem|Swap' | awk '{print $1,$2,$9,$10,$12}'
@psls::top -b -n1 | awk '{print $1,$2,$9,$10,$12}'| tail -n $(top -b -n1 | wc -l | awk '{print $
#date::date
```

รูป 5.3 คอนฟิคกูเรชันไฟล์

คอนฟิคกูเรชันไฟล์มีความสำคัญกับเอเจนท์เป็นอย่างมากเนื่องจากการทำงานจะคล้ายกับการทำงานข้อเราท์เตอร์ที่อ่านสตาร์ทอัพคอนฟิคขึ้นมาเป็นรันนิงคอนฟิค และทำงานตามคอนฟิคดังกล่าว แต่ในระบบนี้สตาร์ทอัพคอนฟิคจะเป็นตัวเดียวกันกับรันนิงคอนฟิค หากต้องการแก้ไขคอนฟิคให้ก็อปปี้ข้อมูลจากไฟล์ดังกล่าวมาใส่ไฟล์สำรองทำการเปลี่ยนแปลงให้เรียบร้อย ตรวจสอบความถูกต้องแล้วจึงใส่ลงไปในไฟล์คอนฟิคจริง ซึ่งเมื่อใส่ลงไปแล้วเอเจนท์จะเปลี่ยนการทำงานมาทำงานตามข้อมูลที่ใส่ลงไปล่าสุดทันที โดยไม่ต้องรีโหลดการทำงานของเอเจนท์

กฎ และข้อแนะนำในการคอนฟิคไฟล์มีดังนี้

1) การใส่ข้อมูลแต่ละบรรทัดจะประกอบด้วยคีย์คั่นด้วย “::” (double colon) ตามด้วยตัวเลข หรือ คำสั่งที่ต้องการ
2) เครื่องหมาย ‘$’ ใส่ไว้หน้าสุดเพื่อบ่งบอกว่าบรรทัดนั้นเป็นการดึงข้อมูลประเภทตัวเลขเท่านั้น
3) เครื่องหมาย ‘@’ใส่ไว้หน้าสุดเพื่อบ่งบอกว่าบรรทัดนั้นเป็นการดึงข้อมูลประเภทสตริงข้อมูลเท่านั้น
4) เครื่องหมาย ‘#’ ใส่ไว้หน้าสุดเพื่อบ่งบอกว่าบรรทัดนั้นเป็นการคอมเมนต์ซึ่งโปรแกรมจะไม่สนใจคำสั่งบรรทัดนี้
5) เครื่องหมาย ‘!’ ไม่สามารถเพิ่มเติมลงไปในคอนฟิคไฟล์ได้ ให้ใช้บรรทัดที่เป็นดีฟอลท์ให้มาอยู่แล้ว และแก้ไขได้เพียงข้อมูลข้างหลังเครื่องหมาย “::” ตามหลักเท่านั้น

6) บรรทัดที่เป็นเครื่องหมาย '$' หรือ '@' เราสามารถเขียนโปรแกรมการดึงข้อมูลแยกออกจากคอนฟิคไฟล์ และใส่คอมมานด์หลัง เครื่องหมาย "::" ได้ เช่น โปรแกรมภาษา shell temp.sh จะใส่ในบรรทัดเป็น $tempkey::bash /path/temp เป็นต้น (ข้อแนะนำ : ควรทดสอบการรันโปรแกรมดังกล่าวกับ ``(back quote) ของภาษาเพิร์ล ก่อนที่จะทำโปรแกรมนั้นมาใช้จริงในคอนฟิคกูเรชันไฟล์)

หากไม่ปฏิบัติตามกฎหรือข้อแนะนำดังกล่าวในการคอนฟิค จะทำให้เอเจนท์ปฏิเสธการทำงานตั้งแต่เริ่มการทำงาน

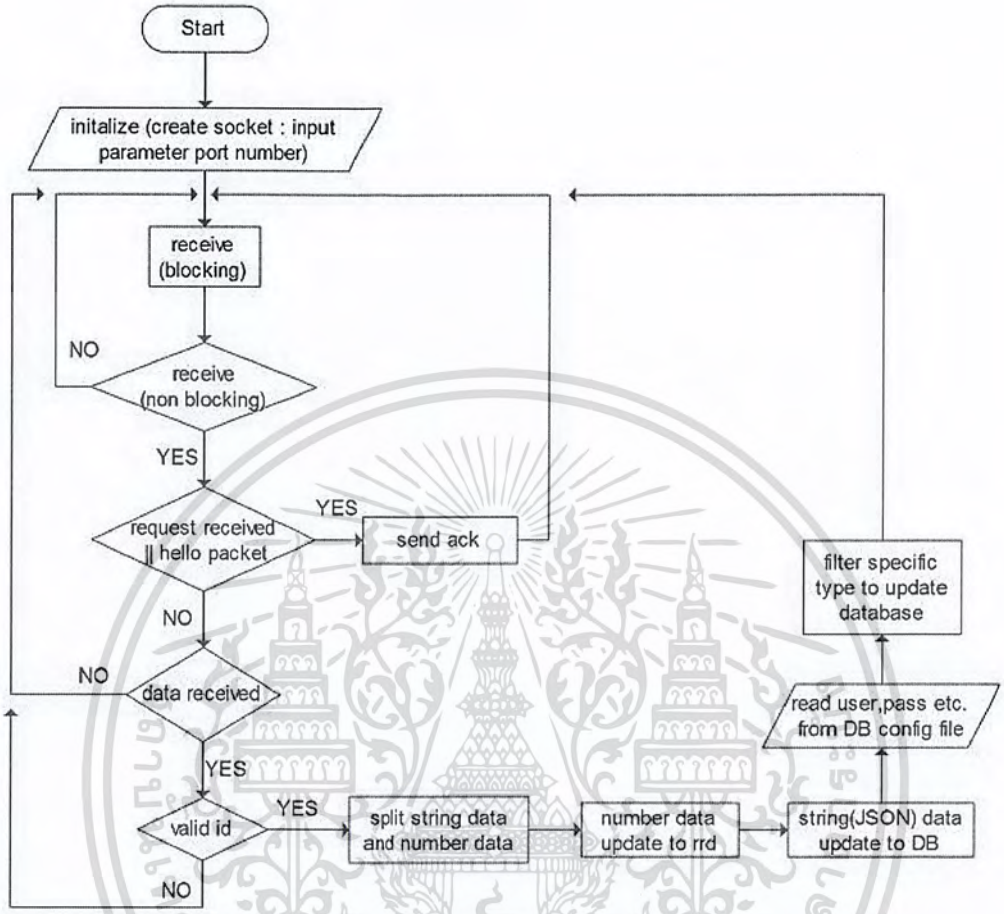
5.3.2 สถิติไฟล์

```
1  usedHdd=28238=2054631.55634244
2  userCPU=28238=0.511939230823723
3  pUsedHdd=28238=56
4  usedMem=28238=319814.776259009
5  apacheload=28238=0.0528874837098942
6  totalAccess=28238=56752.7255117201
7  load1=28238=0.185411856363752
8  freeMem=28238=189511.891918679
9  freeHdd=28238=1637664.44365756
10 totalTraffic=28238=70.7111799702499
11 apacheReq=28238=0.454443559388041
12 sysCPU=28238=2.02057971527748
13 load5=28238=0.180274806997684
14 load15=28238=0.344223386925434
15
```

รูป 5.4 สถิติไฟล์

สถิติไฟล์นั้นเป็นไฟล์ที่เอเจนท์เป็นตัวสร้างขึ้นมาเองขณะทำงาน เราไม่จำเป็นต้องไปคอนฟิคไฟล์ดังกล่าว

5.4 ขั้นตอนการทำงานของเมเนเจอร์



รูป 5.5 การทำงานของเมเนเจอร์โพรเซส

ขั้นตอนการทำงานของเมเนเจอร์จะต้องเรียบง่ายไม่ซับซ้อนเนื่องจากต้องเป็นตัวรับภาระจากข้อมูลที่ส่งมาจากเอเจนท์หลายตัว เริ่มแรกเรากำหนดหมายเลขพอร์ตได้ทางพารามิเตอร์ของคอม - มานด์ที่เราพิมพ์เพื่อรันเมเนเจอร์ได้เลย เมื่อเมเจอร์สตาร์ทขึ้นมาจะทำการสร้างซ็อคเก็ตเปิดพอร์ตรอไว้ และทำการรอรับแพ็คเก็ตที่จะเข้ามาตลอดเวลาโดยที่การรับนี้จะอยู่ในโหมดบล็อกกิ้ง หากไม่มีแพ็คเก็ตเข้ามาก็จะยังคงรออยู่อย่างนั้น

เมื่อได้รับแล้วก็จะดูว่าเป็นแพ็คเก็ตประเภทคอนโทรลคือพวกตรวจสอบสถานะว่ายังอยู่หรือไม่ เป็นต้น ถ้าเป็นแพ็คเก็ตประเภทนี้ เมเนเจอร์ก็จะทำการส่ง “แอ็ค(ack)” กลับไป แต่ถ้าเป็นแพ็คเก็ตที่เป็นข้อมูล เมเจอร์ก็จะทำการตรวจสอบว่าแพ็คเก็ตนั้นมีไอดีที่ถูกต้องหรือไม่ โดยตรวจสอบจากฐานข้อมูล เมื่อผ่านกระบวนการตรวจสอบแล้ว ก็จะนำข้อมูลเหล่านั้นมาแยกประเภทระหว่างข้อมูลที่เป็นตัวเลข และข้อมูลที่เป็นสตริง (ที่ทำการ encode มาจากเอเจนท์แล้ว) ในส่วนที่เป็นข้อมูลตัวเลข เมเนเจอร์จะทำการอัพเดทข้อมูลลงไฟล์ rrd ที่เป็นไฟล์ที่จะเป็นตัวสร้างกราฟเพื่อนำไป

แสดงบนหน้าเว็บนั่นเอง ส่วนข้อมูลที่เป็นสตริง(JSON) นั้นจะถูกเก็บลงในฐานข้อมูล MySQL เพื่อนำข้อมูลเหล่านั้นไปแสดงผลบนเว็บเช่นเดียวกัน

5.5 ระบบเฝ้าระวัง โดยแบ่งตามชั้น

การเฝ้าระวังข้อมูลต่างๆของเซิร์ฟเวอร์แบ่งเป็นชั้นหลัก 3 ชั้นดังนี้

ตาราง 5.1 ชั้นการเฝ้าระวัง

แอปพลิเคชั่น	จำนวนการเข้าถึงแอปพลิเคชั่น, ปริมาณการให้บริการของแอปพลิเคชั่น, ค่าเฉลี่ยโพรเซสของแอปพลิเคชั่นที่กำลังทำงานอยู่ , ความสามารถในการให้บริการของแอปพลิเคชั่น, ความสามารถในการตอบสนองต่อการติดต่อในเครือข่าย
ซอฟต์แวร์ระบบ	เซอร์วิสที่สนใจ, โพรเซสที่กำลังทำงานอยู่
ฮาร์ดแวร์	ซีพียู, หน่วยความจำ, สถานะฮาร์ดดิสก์

ตาราง 5.2 รายการข้อมูลมอนิเตอร์

เฝ้าระวัง	รายละเอียด
ซีพียู	ค่าการใช้งานซีพียูของเซิร์ฟเวอร์
ฮาร์ดดิสก์	ปริมาณการใช้งานฮาร์ดดิสก์ของเซิร์ฟเวอร์
หน่วยความจำ	ปริมาณการใช้งานหน่วยความจำของเซิร์ฟเวอร์
ค่าเฉลี่ยของงานรันอยู่	ค่าเฉลี่ยของงานที่กระตือรือร้นทำงานอยู่ในขณะนั้น
โพรเซส	รายละเอียดโพรเซสที่ทำงานอยู่บนเซิร์ฟเวอร์
ค่าเฉลี่ยของงานรันอยู่ของเว็บเซิร์ฟเวอร์	ค่าเฉลี่ยของงานที่กระตือรือร้นทำงานอยู่ในขณะนั้นของเว็บเซิร์ฟเวอร์
การจราจรเว็บเซิร์ฟเวอร์	ค่าปริมาณการเข้าออกเว็บไซต์
การร้องขอเว็บเซิร์ฟเวอร์	ค่าปริมาณการร้องขอเว็บเซิร์ฟเวอร์ต่อวินาที
การเชื่อมต่อเว็บเซิร์ฟเวอร์	ค่าปริมาณการเชื่อมต่อเว็บเซิร์ฟเวอร์ทั้งหมด
อัพไทม์เว็บเซิร์ฟเวอร์	ค่าจำนวนระยะเวลาตั้งแต่ทำการเปิดเครื่องเซิร์ฟเวอร์จนถึงปัจจุบัน
เน็ตเวิร์คอินเตอร์เฟส	สถานะเชื่อมต่อของเซิร์ฟเวอร์กับระบบเครือข่าย
เซอร์วิส	สถานะของเซอร์วิสที่สนใจ

5.6 การเตือนภัย

ระบบเฝ้าระวังนั้น เมื่อเกิดเหตุการณ์ที่ผิดปกติ ก่อนที่จะมีการแก้ไขได้ ต้องมีการเตือนภัยเกิดขึ้นก่อน ในส่วนของการเตือนภัยสำหรับระบบเฝ้าระวังนี้ จะทำการแจ้งเตือนแก่ผู้ดูแลระบบผ่านทางอีเมล์เมื่อเกิดเหตุการณ์ที่ตรงตามเงื่อนไขการเตือนภัยในขั้นต่างๆ โดยแบ่งขั้นการเตือนภัยออกเป็น 3 ขั้น ดังนี้ ขั้นที่หนึ่ง เตือนภัยขั้นปกติจะไม่มีการแจ้งเตือนภัยใดๆ เพราะถือว่าระบบอยู่ในสถานะปกติ ไม่มีความเสี่ยง ขั้นที่สอง เตือนภัยขั้นกลางเป็นขั้นเตือนภัยที่รุนแรงปานกลาง ระบบเฝ้าระวังจะทำการส่งอีเมล์แจ้งแก่ผู้ดูแลระบบเพื่อเตือนให้ผู้ดูแลระบบรับรู้ ขั้นสุดท้าย เตือนภัยขั้นวิกฤต จะทำการส่งอีเมล์แจ้งเตือนไปยังผู้ดูแลระบบว่าเกิดเหตุการณ์ตามที่ได้ตามที่ไว้กำหนดเงื่อนไข โดยขั้นสุดท้ายนี้เป็นขั้นที่รุนแรงที่สุดซึ่งอาจทำให้ระบบมีปัญหาได้หากไม่รีบแก้ไขปัญหาและอาจเกิดการทำแอคชันบางประการเพื่อช่วยเหลือเบื้องต้นแก่ระบบ การออกแบบการเตือนภัย มีดังนี้

ตาราง 5.3 รายละเอียดการเตือนภัย

ตัวเฝ้าระวัง	การเตือนภัย		
	เตือนภัยขั้นวิกฤต	เตือนภัยขั้นกลาง	เตือนภัยขั้นปกติ
ซีพียู	มากกว่า 90% เป็นระยะเวลา มากกว่า 3 นาที	ประมาณ 51 – 89%	น้อยกว่า 50%
หน่วยความจำ	มากกว่า 80% เป็นระยะเวลา มากกว่า 3 นาที	ประมาณ 71 – 79%	น้อยกว่า 70%
ฮาร์ดดิสก์ก์	มากกว่า 80% เป็นระยะเวลา มากกว่า 3 นาที	ประมาณ 51 – 79%	น้อยกว่า 50%
ค่าเฉลี่ยของงานที่ทำงานอยู่ในขณะนั้นที่ 5 นาที	มากกว่า 1.80 เป็นระยะเวลา มากกว่า 3 นาที	ประมาณ 1.50 – 1.79	น้อยกว่า 1.00
ค่าเฉลี่ยของงานที่ทำงานอยู่ในขณะนั้นที่ 15 นาที	มากกว่า 1.80 เป็นระยะเวลา มากกว่า 3 นาที	ประมาณ 1.50 – 1.79	น้อยกว่า 1.00

ตาราง 5.3 รายละเอียดการเตือนภัย (ต่อ)

ตัวเฝ้าระวัง	การเตือนภัย		
	เตือนภัยขั้นวิกฤต	เตือนภัยขั้นกลาง	เตือนภัยขั้นปกติ
เซอร์วิส	เซอร์วิสไม่สามารถให้บริการได้ (และได้ทำการรีสตาร์ทมากกว่า 3 ครั้งขึ้นไปแล้ว)		

5.7 การแก้ไขปัญหา

เมื่อมีเหตุการณ์ผิดปกติเกิดขึ้น จะมีการเตือนภัยให้ผู้ดูแลระบบทราบ และโดยปกติแล้วผู้ดูแลระบบจะเป็นผู้แก้ไขปัญหา ซึ่งปัญหาบางประเภทเราสามารถกำหนดให้เอเจนท์แก้ไขปัญหาแทนได้ เช่นโพรเซสไม่ตอบสนอง เอเจนท์ก็จะสั่งให้ทำลาย และเริ่มโพรเซสนั้นใหม่อีกครั้ง เป็นต้น ซึ่งเหตุการณ์ที่เกิดขึ้น และการแก้ไขปัญหามีดังนี้

ตาราง 5.4 รายละเอียดการแก้ปัญหา

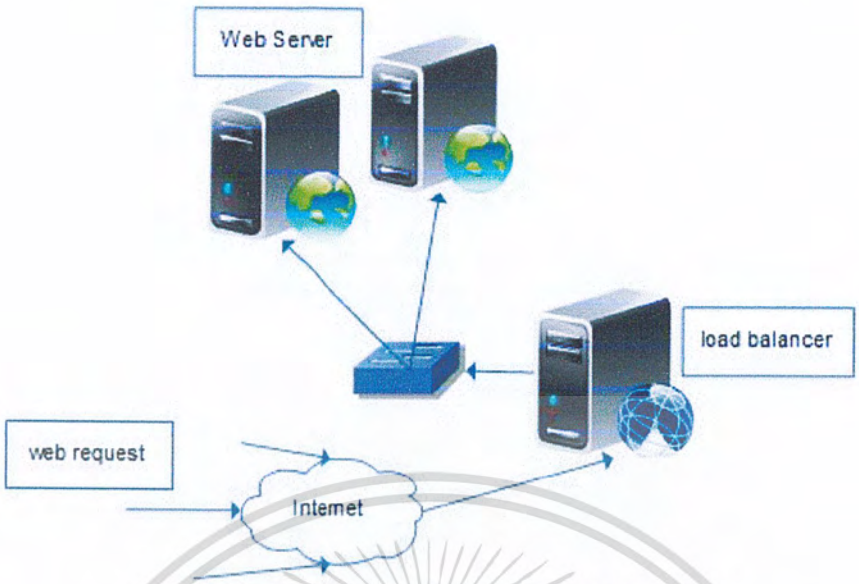
เฝ้าระวัง	เหตุการณ์	การแก้ปัญหา
โพรเซส	ไม่ตอบสนองต่อการทำงาน	สั่งทำลายโพรเซส
โพรเซส	โพรเซสมากกว่า 3 ตัวขึ้นไป ไม่ตอบสนองต่อการทำงาน	สั่งรีสตาร์ทเครื่องเซิร์ฟเวอร์
ซีพียู	ค่าเปอร์เซ็นต์ของซีพียูอยู่ในช่วงวิกฤตเป็นเวลามากกว่า 5 นาที	เกิดการอพยพโพรเซสด้วยอัลกอริทึม Round – Robin
ค่าเฉลี่ยการทำงานของซีพียูที่ 5 นาที	ค่าเฉลี่ยของซีพียูอยู่ในช่วงวิกฤตเป็นเวลามากกว่า 3 นาที	เกิดการอพยพโพรเซสด้วยอัลกอริทึม Round – Robin
ค่าเฉลี่ยการทำงานของซีพียูที่ 15 นาที	ค่าเฉลี่ยของซีพียูอยู่ในช่วงวิกฤตเป็นเวลามากกว่า 5 นาที	เกิดการอพยพโพรเซสด้วยอัลกอริทึม Least – Connection

ตาราง 5.4 รายละเอียดการแก้ปัญหา (ต่อ)

เฝ้าระวัง	เหตุการณ์	การแก้ปัญหา
ค่าเฉลี่ยการทำงานของซีพียูที่ 15 นาที	ค่าเฉลี่ยของซีพียูอยู่ในช่วงวิกฤตเป็นเวลามากกว่า 10 นาที	เกิดการอพยพโพรเซสด้วยอัลกอริทึม Least - Connection และแจ้งเตือนขั้นวิกฤตผ่านทางอีเมล์
เซอร์วิสการให้บริการเว็บเซิร์ฟเวอร์	ไม่ตอบสนองต่อการทำงานหรือไม่สามารถให้บริการได้	สั่งรีสตาร์ทเซอร์วิส

5.8 การอพยพโพรเซส

ปัญหาของการรับภาระหนักของเครื่องเซิร์ฟเวอร์อันเกิดจากการใช้ทรัพยากรของเครื่องเซิร์ฟเวอร์มากเกินไปจนบางครั้งไม่สามารถให้บริการได้ตามปกติ ซึ่งในที่นี้คือเว็บเซิร์ฟเวอร์ หากมีการใช้งานเว็บจากหลากหลายยูเซอร์เป็นเวลานาน เว็บมีการเก็บข้อมูลลงบนเซิร์ฟเวอร์โดยไม่กำจัดทิ้งเป็นเวลานานอาจทำให้บางไดเร็กทอรีเต็ม แคชเต็ม หรืออีกกรณีหนึ่งคือคำร้องที่เข้ามามากเกินไป ทรัพยากรของเครื่อง ณ ขณะนั้นไม่สามารถรองรับได้ คำร้องที่เข้ามาหลังจากระบบมีปัญหาก็จะไม่สามารถเข้าใช้บริการหรือได้รับบริการที่ช้าลง และส่งผลต่อโพรเซสอื่นๆที่อยู่บนเครื่องเซิร์ฟเวอร์นั้น และที่สำคัญยิ่งคือส่งผลต่อความน่าเชื่อถือในการเข้าใช้บริการอีกด้วย ปัญหาเหล่านั้นเราจำแนกจากโพรเซสที่ต้องทำการแก้ไข จึงจำแนกเป็นสองกรณีคือ กรณีที่เอเจนท์โพรเซสสามารถแก้ไขเองได้ภายในเครื่องที่มันรันอยู่ ส่วนอีกกรณีคือกรณีที่ต้องจัดการคำร้องที่เข้ามาว่าจะให้เครื่องไหนมารับคำร้องนั้น ซึ่งในกรณีนี้จะเป็นหน้าที่ของเมเนเจอร์เป็นตัวจัดการไม่สามารถใช้เอเจนท์แก้ปัญหานี้ได้ ซึ่งเครื่องที่เมเนเจอร์โพรเซสทำงานอยู่จะทำหน้าที่เป็นตัวจัดงานให้สมดุล (Load balancer) อีกด้วย เพราะคำร้องที่เข้ามาจะต้องผ่านเซิร์ฟเวอร์ที่เป็นตัวจัดงานให้สมดุลเสมอดังรูป



รูป 5.6 ลักษณะคำร้องที่เข้ามาที่เว็บเซิร์ฟเวอร์

เมเนเจอร์จะทำงานร่วมกับโปรแกรมชื่อ Crossroad ซึ่งเป็นเครื่องช่วยในเรื่องของการจัดงานให้สมดุล Crossroad มีอัลกอริทึมที่ใช้ทำการกระจายโหลดมากกว่าหนึ่งอัลกอริทึมซึ่งเราสามารถเลือกใช้ได้ผ่านการพิมพ์คอมมานด์ซึ่งช่องทางนี้จะเป็นช่องทางให้เมเนเจอร์ทำงานร่วมกับ Crossroad ได้ นั่นคือเมเนเจอร์จะไปเรียกคำสั่งของ Crossroad ตามเงื่อนไขที่กำหนดต่อไป

บทที่ 6
เครื่องมือช่วยในการพัฒนา ระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซส

6.1 การจัดสรรงานให้สมดุล

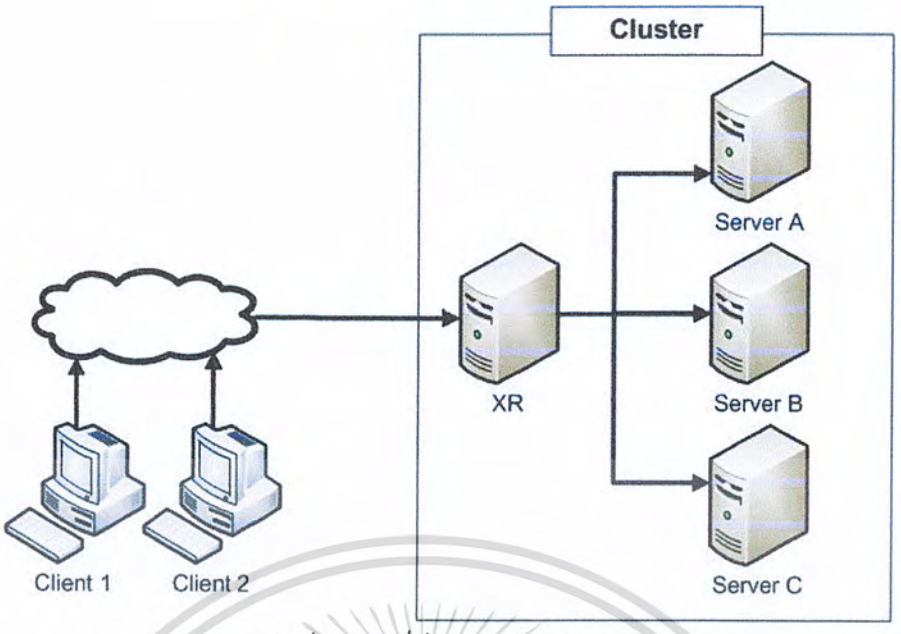
การจัดสรรงานให้สมดุล (Load Balancer) เป็นเทคนิคการจัดสรรงานให้สมดุลอย่างสม่ำเสมอ เพื่อให้ได้รับประสิทธิภาพการประมวลผลรวมสูงที่สุด โดยสามารถจัดสรรงานระหว่างเครื่องประมวลผลให้เหมาะสมก่อนเริ่มการประมวลผล เพื่อสร้างสมดุลของโหลดระหว่างหน่วยประมวลผล ทำให้ได้รับการใช้ทรัพยากรที่ดีที่สุด สามารถรับงานที่เข้ามาได้มากขึ้น ใช้เวลาการตอบสนองลดลง และหลีกเลี่ยงการทำงานเกินพิกัดจนทำให้ระบบไม่สามารถให้บริการได้

6.2 XR Crossroads Load Balancer and Fail Over Utility

XR คือโค้ดการจัดสรรงานให้สมดุล และ Fail Over สำหรับบริการที่ทำงานบนโปรโตคอลทีซีพีแบบเปิดเผย โดยมีคุณสมบัติครอบคลุมโดยยูสเซอร์สามารถกำหนดค่าการคอนฟิคเองได้ XR สามารถสั่งรีสตาร์ทเซิร์ฟเวอร์เมื่อเซิร์ฟเวอร์ไม่ทำงาน สามารถรายงานแสดงสถานะของระบบ มีอัลกอริทึมการกระจาย (Dispatching Algorithm) ให้เลือกหลากหลาย และสามารถกำหนดมีอัลกอริทึมการกระจาย ในกรณีพิเศษได้ เป็นต้น XR สามารถใช้ได้กับบริการที่ทำงานอยู่บนโปรโตคอลทีซีพี เช่น HTTP, SMTP, SSH เป็นต้น ในกรณีบริการแบบ HTTP จะทำการจัดสรรงานให้สมดุลเมื่อมีไคลเอนต์หลายๆ ไคลเอนต์ร้องขอบริการ นอกจากนี้ยังมีคุณสมบัติของ Fail Over ถ้าเซิร์ฟเวอร์เครื่องใดเครื่องหนึ่งไม่ทำงาน XR จะค้นหาเส้นทางที่จะส่งการร้องขอของไคลเอนต์ไปยังเซิร์ฟเวอร์เครื่องอื่นแทน โดยไคลเอนต์จะไม่ทราบว่ามีช่วงเวลาที่ระบบเซิร์ฟเวอร์ไม่ทำงาน และยังสามารถจัดการข้อมูลผ่าน Web Interface ได้อีกด้วย

6.2.1 องค์ประกอบของ XR

เซิร์ฟเวอร์ที่ให้บริการทั้งหมดมองเป็นกลุ่มเดียวกันเรียกว่าคลัสเตอร์ โดยมีเซิร์ฟเวอร์หนึ่งเครื่องได้ทำการติดตั้ง XR มีหน้าที่รับการร้องขอบริการที่เข้ามาจากไคลเอนต์ แล้วทำการจัดสรรงานให้สมดุลไปยังเซิร์ฟเวอร์ที่ให้บริการ



รูป 6.1 องค์ประกอบของ XR

6.2.2 อัลกอริทึมการกระจาย

XR จะทำการจัดสรรงานให้สมดุลไปยังเซิร์ฟเวอร์เครื่องต่างๆที่อยู่ภายในกลุ่มคลัสเตอร์ โดยการส่งข้อมูลเป็นไปตามการเชื่อมต่อแบบโปรโตคอลทีซีพี และมีกฎเกณฑ์อัลกอริทึมการกระจาย ดังนี้

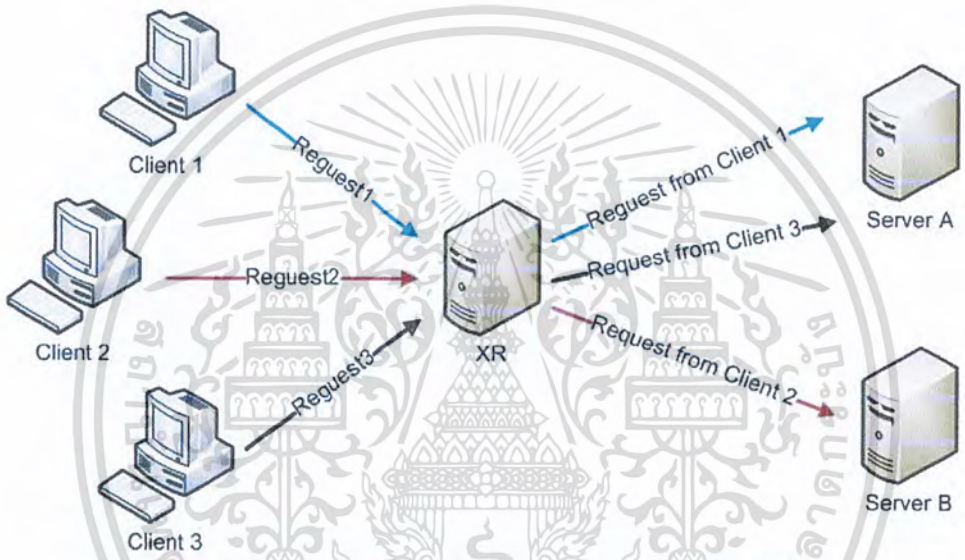
1) Round-Robin
2) Least-Connection
3) First-Available
4) External
5) Strict-Hashed-IP
6) Lax-Hashed-IP
7) Strict-Stored-IP
8) Lax-Stored-IP
9) Weighted-Load

เทคนิค Round-Robin เป็นหนึ่งในหลายเทคนิคที่นิยมใช้ในการจัดสรรงานให้สมดุลแบบหนึ่ง โดยมีหลักการว่า ชื่อโดเมนของเซิร์ฟเวอร์หนึ่งชื่อ สามารถมีหมายเลขไอพีที่แตกต่างกันได้ในแต่ละครั้งที่มีการร้องขอชื่อโดเมนเดียวกัน โดยโดเมนเนมเซิร์ฟเวอร์จะวนรอบส่งหมายเลขไอพีที่แตกต่างกันให้กับการร้องขอชื่อโดเมนเดียวกัน ทำให้ไคลเอนต์ไม่ทราบว่ากำลังติดต่อกับเซิร์ฟเวอร์ใด เช่น มีเซิร์ฟเวอร์ X Y Z แล้วมีการร้องขอจากไคลเอนต์ A จะส่งคำร้องขอไปยังเซิร์ฟเวอร์ X มีการร้องขอจากไคลเอนต์ B จะส่งคำร้องขอไปยังเซิร์ฟเวอร์ Y มีการร้องขอจากไคลเอนต์ C จะส่งคำ

ร้องขอไปยังเซิร์ฟเวอร์ Z และเมื่อมีการร้องขอจากไคลเอนต์ D จะส่งคำร้องวนกลับไปยังเซิร์ฟเวอร์ X เป็นต้น ดังนั้น Round Robin จึงเป็นการกระจายภาระงานให้สมดุลแบบวนรอบ

Least-Connection เป็นเทคนิคการจัดสรรงานให้สมดุล โดยมีหลักการว่า เซิร์ฟเวอร์ที่ถูกเชื่อมต่อ หรือมีการเชื่อมต่ออยู่ในขณะนั้นน้อยที่สุด จะถูกให้รับการเชื่อมต่อจากการร้องขอของไคลเอนต์ โดยอัลกอรึทึมแบบ Least-Connection จะมีการนับจำนวนการเชื่อมต่อกับเซิร์ฟเวอร์แต่ละตัวในขณะนั้น ดังนั้นเซิร์ฟเวอร์ที่มีการเชื่อมต่อน้อยสุดจะถูกเลือกให้รับการร้องขอจากไคลเอนต์

First-Available เป็นเทคนิคการจัดสรรงานให้สมดุล โดยมีหลักการว่า เซิร์ฟเวอร์ตัวใดพร้อมทำงานจะถูกให้รับการร้องขอจากไคลเอนต์ทันที เป็นต้น



รูป 6.2 การจัดสรรงานให้สมดุลของ XR แบบ Round-Robin

6.3 Round Robin Database Tool

Round Robin Database Tool (RRDTool) พัฒนาโดย Tobias Oetiker เป็นเครื่องมือสำหรับเก็บบันทึกข้อมูล (Data Logging) และวิเคราะห์ข้อมูล ซึ่งรวบรวมมาจากแหล่งข้อมูล (Data Source) ทุกชนิด โดยวิเคราะห์ข้อมูลแล้วแสดงออกมาในรูปกราฟแสดงเวลา โดยแกน X ของกราฟแสดงค่าเวลา ส่วนแกน Y ของกราฟแสดงค่าของตัวแปร โดยข้อมูลที่แสดงอยู่ในกราฟมาจากฐานข้อมูลใน RRDTool ซึ่งความแตกต่างระหว่าง RRDTool กับฐานข้อมูลอื่นๆมีดังนี้

1) ฐานข้อมูลของ RRDTool ทำการจัดเก็บข้อมูลไว้ในฐานข้อมูล และแสดงข้อมูลด้วยคำสั่งสร้างกราฟเพื่อแสดงผล ซึ่งฐานข้อมูลอื่นๆมีหน้าที่ในการจัดเก็บข้อมูลเท่านั้น ไม่มีการสร้างกราฟแสดงผล
2) ในกรณีฐานข้อมูลเชิงเส้น เมื่อมีข้อมูลเข้ามาใหม่ ข้อมูลเหล่านั้นจะถูกจัดเก็บที่ด้านล่างของตารางฐานข้อมูลส่งผลให้ขนาดตารางข้อมูลเพิ่มขึ้นเรื่อยๆ ในขณะที่ขนาดฐานข้อมูลของ

RRDTool จะพิจารณาจากเวลาที่ต้องการสร้างกราฟ เสมือนว่าฐานข้อมูลเป็นลักษณะวงกลม ข้อมูลจะเพิ่มรอบๆวงกลม เมื่อข้อมูลถูกเพิ่มจนถึงจุดเริ่มต้น จะทำการเขียนทับข้อมูลเดิม ส่งผลให้ขนาดของฐานข้อมูลคงที่ จึงเป็นเหตุผลของชื่อ Round Robin

3) ฐานข้อมูลอื่นๆทำหน้าที่จัดเก็บข้อมูลที่ได้มา แต่ RRDTool สามารถคำนวณอัตราการเปลี่ยนแปลงของค่าจากก่อนหน้าถึงปัจจุบัน และจัดเก็บข้อมูลเหล่านี้แทน

4) ฐานข้อมูลอื่นๆจะอัพเดตเมื่อได้รับค่า แต่ RRDTool เป็นโครงสร้างที่ต้องการรับค่าในช่วงเวลาที่กำหนดไว้ล่วงหน้าแล้ว หากไม่ได้รับค่าตามช่วงเวลาที่กำหนด จะทำการเก็บค่าที่ไม่รู้จักไว้ในช่วงเวลานั้นแทน

6.4 โครงสร้างของ Round Robin Database

โครงสร้างของฐานข้อมูล RRD แตกต่างจากฐานข้อมูลเชิงเส้นอื่นๆ ฐานข้อมูลอื่นๆจะกำหนดตารางด้วยคอลัมน์ และมีพารามิเตอร์อื่นๆอีกมากมาย การกำหนดข้อมูลต่างๆเหล่านี้มีความซับซ้อนอย่างมากโดยเฉพาะอย่างยิ่งในฐานข้อมูลขนาดใหญ่ ส่วนฐานข้อมูลของ RRDTool เน้นการใช้งานในการเฝ้าระวังระบบ และมีโครงสร้างข้อมูลที่ไม่ซับซ้อน พารามิเตอร์ในฐานข้อมูล RRD กำหนดเพื่อเป็นตัวแปรสำหรับค่า และเก็บค่าข้อมูลเหล่านี้ไว้ โดยคำศัพท์ที่เกียวข้องกับฐานข้อมูล RRDTool ประกอบด้วย Data Source (DS), Date Source Type (DST), Round Robin Archive (RRA), Consolidation Function (CF) เป็นต้น

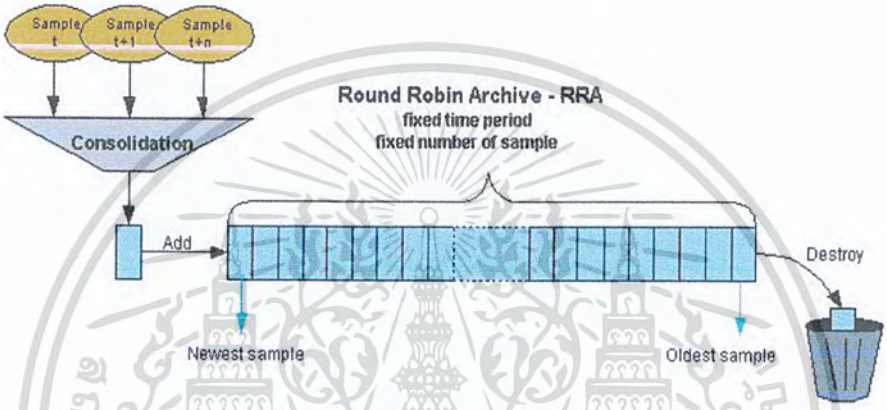
1) Data Source มีตัวแปรพารามิเตอร์ซึ่งจะถูกบันทึกไว้ในฐานข้อมูล โดยสามารถเก็บข้อมูล DS ได้มากมายไว้ในฐานข้อมูลได้ตามที่ต้องการ โดยค่าใหม่ของ DS จะถูกเพิ่มข้อมูลลงในฐานข้อมูล โดยเรียกค่าใหม่นี้ว่า Primary Data Point (PDP)

2) Date Source Type ค่าอาทิวเมนต์ของแหล่งข้อมูลจะขึ้นอยู่กับการกำหนดประเภทของ DST โดยประเภทของ DST ประกอบด้วย GAUGE COUNTER DERIVE ABSOLUTE และ COMPUTE เป็นต้น

3) Round Robin Archive ข้อมูลที่ต้องการรวบรวมข้อมูลเข้าด้วยกันจะถูกเก็บไว้ที่ RRA การทำ RRA ทำให้การจัดเก็บข้อมูลในช่วงเวลาหนึ่งมีประสิทธิภาพมากขึ้น และทำให้พื้นที่จัดเก็บข้อมูลคงที่อยู่เสมอ สาเหตุมาจาก ถ้าต้องการจัดเก็บข้อมูลจำนวน 1000 ข้อมูลในช่วงระยะเวลา 5 นาที RRDTool จะทำการจองพื้นที่เพื่อจัดเก็บข้อมูลจำนวน 1000 ข้อมูล และจองพื้นที่สำหรับส่วนหัว (Header) ของข้อมูล ในส่วนหัวของข้อมูลจะเก็บตัวแปรชนิดพอยเตอร์ เพื่อชี้สล็อต (Slot) ในพื้นที่จัดเก็บข้อมูล (Storage) ที่ถูกเขียนล่าสุด เมื่อมีข้อมูลใหม่ล่าสุดเข้ามาจะถูกเขียนลงไปใน RRA ถัดจากตัวที่ถูกเขียนล่าสุด และยังสามารถกำหนด

หลายๆ RRAในฐานข้อมูล RRD เดียวกันได้ โดย RRA จะไม่ทำให้ฐานข้อมูล RRD เพิ่มขึ้น และข้อมูลชุดเก่าจะถูกกำจัดออกไปโดยอัตโนมัติ

4) Consolidation Function เป็นฟังก์ชันรวบรวมข้อมูลเข้าด้วยกัน (Consolidation) ทำให้สามารถเก็บรักษาข้อมูลได้เป็นระยะเวลานาน โดยสามารถทำการรวมข้อมูล PDP ผ่านฟังค์ชันรวบรวมข้อมูลแบบต่างๆ เช่น AVERAGE MIN MAX LAST โดย AVERAGE เก็บข้อมูลค่าเฉลี่ยของ PDP ไว้ ส่วน MIN เก็บข้อมูลค่าที่น้อยที่สุดของ PDP ต่อไป MAX เก็บข้อมูลค่าที่มากที่สุดของ PDP และ LAST เก็บข้อมูลล่าสุดของ PDP ไว้



รูป 6.3 การทำงานของ Round Robin Database

6.4.1 การสร้าง Round Robin Database

ฟังก์ชัน Create ใน RRDTool เป็นการสร้างไฟล์ Round Robin Database (RRD) ขึ้นมา โดยไฟล์ที่ถูกสร้างประกอบด้วยขนาดของข้อมูลที่ต้องการเก็บทั้งหมด และในแต่ละฟิลด์ของข้อมูลจะเป็นประเภท Unknow เนื่องจากยังไม่มีข้อมูลเก็บอยู่

RRD จะเก็บข้อมูลไว้ใน Round Robin Archive (RRA) โดย RRA สามารถเก็บจำนวนของค่าข้อมูล หรือเก็บค่าทางสถิติของแต่ละแหล่งข้อมูลที่กำหนดได้ เมื่อข้อมูลถูกป้อนลงใน RRD ข้อมูลจะมีขนาดพอดีกับช่วงเวลาของความยาวที่กำหนดไว้ แล้วเรียกข้อมูลเหล่านี้ว่า Primary Data Point (PDP)

ข้อมูลจะถูกจัดเก็บด้วยฟังก์ชันรวบรวมข้อมูล (Consolidation Function) ซึ่งสามารถมีการรวบรวมข้อมูลได้หลายๆฟังก์ชัน โดยสามารถทำการรวมข้อมูล PDP ผ่านฟังก์ชันรวบรวมข้อมูลแบบต่างๆ เช่น AVERAGE MIN MAX LAST เป็นต้น

```
$command = 'rrdtool create ../rrd_/'.$ip.'.rrd --step 60 \
            DS:userCPU:GAUGE:120:U:U \
            DS:sysCPU:GAUGE:120:U:U \
            DS:load_1:GAUGE:120:U:U \
            DS:load_5:GAUGE:120:U:U \
            DS:load_15:GAUGE:120:U:U \
            DS:pUsedHdd:GAUGE:120:U:U \
            DS:usedHdd:GAUGE:120:U:U \
            DS:freeHdd:GAUGE:120:U:U \
            DS:usedMem:GAUGE:120:U:U \
            DS:freeMem:GAUGE:120:U:U \
            DS:apacheLoad:GAUGE:120:U:U \
            DS:apacheReq:GAUGE:120:U:U \
            DS:totalAccess:GAUGE:120:U:U \
            DS:totalTraffic:GAUGE:120:U:U \
            RRA:MIN:0.5:1:1500 \
            RRA:MAX:0.5:1:1500 \
            RRA:AVERAGE:0.5:1:1500 ';
```

รูป 6.4 การสร้าง RRD ใน RRDTool

6.5 โครงสร้างของกราฟใน Round Robin Database Tool

6.5.1 การสร้างกราฟใน Round Robin Database Tool

ฟังก์ชันกราฟของ RRDTool ถูกใช้เพื่อนำเสนอข้อมูลจากฐานข้อมูล RRD โดยแสดงข้อมูลเป็นภาพ และแสดงรายงานเชิงตัวเลข โดยการใช้คำสั่ง Fetch ดึงค่าจากฐานข้อมูล โดยค่าที่ดึงมาจะถูกนำไปพล็อตกราฟตามพารามิเตอร์กำหนดไว้ ซึ่งกราฟหนึ่งสามารถแสดงแหล่งข้อมูลจากฐานข้อมูลที่แตกต่างกันได้ นอกจากนี้ยังสามารถแสดงค่าจากฐานข้อมูลหลายๆฐานข้อมูลมารวมอยู่ในกราฟเดียวกันได้ ซึ่งบ่อยครั้งมีการนำค่าที่ดึงมาจากฐานข้อมูลนำไปดำเนินการทางคณิตศาสตร์ก่อนนำไปพล็อตกราฟ เช่นค่าการใช้หน่วยความจำถูกระบุไว้เป็นหน่วย Kbyte ส่วนเส้นทางการจราจรบนอินเตอร์เฟสถูกระบุไว้เป็นหน่วย Byte ดังนั้นกราฟจะต้องทำการแปลงค่าที่ได้ก่อนถ้าต้องการแสดงค่าในหน่วย Mbytes หรือ mbps แล้วจึงนำมาพล็อตกราฟต่อไป

นอกจากคำสั่งสร้างกราฟใน RRDTool ยังอนุญาตให้ทำการคำนวณแปลงหน่วยแล้ว ยังสามารถใช้ตรรกะทางคณิตศาสตร์ เช่น มากกว่า น้อยกกว่า ถ้าแล้ว หรืออื่นๆ เป็นต้น ถ้าฐานข้อมูลหนึ่งประกอบด้วยหลายๆ RRA อาจทำให้เกิดคำถามขึ้นได้ว่า RRDTool มีหลักการเลือก RRA เพื่อดึงค่าอย่างไร RRDTool จะมองหลายสิ่งหลายอย่างในการเลือก RRA ใดๆ อย่างแรกต้องมีสิ่งที่ทำให้แน่ใจว่า RRA ครอบคลุมเท่าๆกับกรอบระยะเวลาของกราฟนั้นๆมากที่สุด อย่างที่สองจะดูที่ความละเอียดของ RRA เปรียบเทียบกับความละเอียดของกราฟ โดยจะเลือกความละเอียดที่เท่ากันหรือสูงกว่าเป็นหลัก

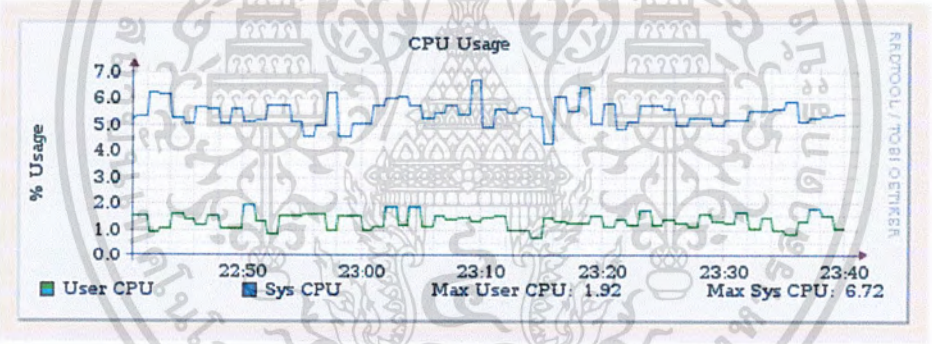
ค่าของตัวแปรที่แตกต่างกันสามารถนำเสนอได้ 5 รูปทรงที่แตกต่างกันในหนึ่งกราฟได้ ประกอบด้วยรูปแบบ AREA, LINE1, LINE2, LINE3, STACK เป็นต้น ในรูปทรง AREA จะแสดงค่าเป็นพื้นสีแบบทึบแสดงถึงขอบเขตของพื้นที่นี้ รูปทรง LINE1, LINE2, LINE3 จะแสดงค่าด้วย

เส้นธรรมดา ซึ่งเส้นนี้สามารถขยายกว้างขึ้นได้ ส่วนรูปทรง STACK แสดงค่าเป็นแบบพื้นที่ส่วนด้านบนของรูปทรง AREA, LINE1, LINE2, LINE3

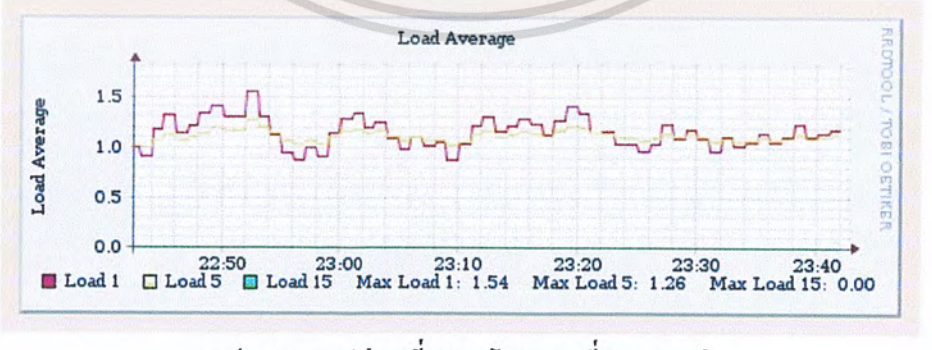
```
$cmdGraphApReq ='rrdtool graph rrd_/imgs/'.$tmpl_uri[2].'-req.png \
                -w 495 -h 167 -a PNG --full-size-mode \
                --end now --start end-3600s \
                --slope-mode \
                --font DEFAULT:7: \
                --title "Apache Request" \
                --vertical-label "Request/Sec" \
                --lower-limit 0 \
                DEF:aReq=rrd_/'.$tmpl_uri[2].'.rrd:apacheReq:MAX \
                LINE1:aReq#ff0000:"Apache Request" \
                GPRINT:aReq:MAX:"Max Request/Sec \: %5.21f" ';
```

รูป 6.5 คำสั่งสร้างกราฟใน RRDTool

6.6 รูปแบบกราฟใน RRDTool



รูป 6.6 กราฟการใช้งานซีพียู



รูป 6.7 กราฟค่าเฉลี่ยของโพรเซสที่ทำงานอยู่

บรรณานุกรม

สานนท์ ฉิมมณี. 2552. **เขียนโปรแกรม และเรียนรู้เครือข่ายคอมพิวเตอร์ด้วย Ubuntu+Perl.** นนทบุรี : ไอดีซีฯ.

บัณฑิต จามรภูติ. 2549. **คู่มือระบบยูนิกซ์ FreeBSD เล่ม 1.** กรุงเทพฯ : Bandhit Press.

Chiu, D. M. and Sudama, R. 1992. **Network Monitoring Explained**. West Sussex : Ellis Horwood Limited.

Behrouz, A.F. and Richard, F.G. 2003. **Unix and Shell Programming.** California : Brooks/Cole.

Tim, M. 2007. **Minimal Perl for Unix and Linux People.** New York : Manning Publication Co.

Suppi, B.R. 2007. **GNU/Linux advanced Administration.** Barcelona : Eureca Media.

Mark, M. Jeffrey, O. and Alex, S. 2001. **Advanced Linux Programming.** Indiana : New Riders Publishing.

Justin David, S. 2003. "**Fault Tolerance using Whole-Process Migration and Speculative Execution.**" Master Degree Thesis of California Institute of Technology.

Dejan, S. Fred, D. Yves, P. and Songnian, Z. 2000. "Process Migration." **ACM Computing Survey.** 32(3) : 241-299.

Ramon Lawrence. 1998. "**A Survey of Process Migration Mechanisms.**" Department of Computer Science University of Manitoba

Asad, H. Khanh, N. Paul, N. and Simon Turner. "**Process Migration for Load Balancing and Fault Tolerance.**" Dept. of Computer Science and Engineering University of California

ภัทระ เกียรติเสวี. 2542. **การสื่อสารข้อมูลแบบ Connection-Oriented และ Connectionless.** [Online]. Available : http://www.sa.ac.th/e-learning/internet/connection.html

พรเทพ ศรีภักตรา. 2552. **แบบจำลอง TCP/IP.** [Online]. Available : http://portal.in.th/eleccom53/pages/6029/

สารานุกรมเสรี. 2553. **Internet Protocol Suite.** [Online]. Available : http://th.wikipedia.org/wiki/Internet_protocol_suite

สารานุกรมเสรี. 2553. **User Datagram Protocol.** [Online]. Available : http://th.wikipedia.org/wiki/User_Datagram_Protocol

Nixcraft. 2004. **Monitoring.** [Online]. Available : http://bash.cyberciti.biz/shell/monitoring/

Linuxinsight Administrator. 2006. **The /proc filesystem documentation.** [Online]. Available : http://www.linuxinsight.com/proc_filesystem.html

Script Hat Administrator. 2009. **Simple perl mailer script.** [Online]. Available : http://scripthat.com/viewtopic.php

Sontaya. 2009. **Kill Process.** [Online]. Available : http://www.susethailand.com/suseforum/index.php?topic=587.0

Arslan. 2010. **Shell Scripts To Monitor Swap and Hard Disk Usage.** [Online]. Available : http://blogs.oranix.com/?p=112

Karel Kubat. 2008. **XR: Crossroads Load Balancer and Fail Over Utility.** [Online]. Available : http://crossroads.e-tunity.com/documentation.xr

Luteus. 2004. **Introduction to RRD - Round Robin Database.** [Online]. Available : http://www.loriotpro.com/Products/On-line_Documentation_V5/

Tobias Oetiker. 2011. **RRDTool.** [Online].

Available : http://www.mrtg.org/rrdtool/doc/rrdtool.en.html

Tobias Oetiker. 2011. **RRDGraph.** [Online].

Available : http://www.mrtg.org/rrdtool/doc/rrdgraph.en.html

Tobias Oetiker. 2011. **RRDCreate.** [Online].

Available : http://www.mrtg.org/rrdtool/doc/rrdcreate.en.html

Linux. 1998. **Job Scheduling Algorithms in Linux Virtual Server.** [Online].

Available : http://www.linuxvirtualserver.org/docs/scheduling.html

REDHAT. 2010. **LVS Scheduling.** [Online].

Available : http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux

ภาคผนวก ก
ตัวอย่างคำสั่งที่ใช้ในระบบปฏิบัติการลีนุกซ์

ก.1 คำสั่งที่เกี่ยวข้องกับการทำงานของซีพียู และโพรเซส

คำสั่งเหล่านี้จะใช้ในการดูลักษณะการทำงานของซีพียู แสดงรายละเอียดของโพรเซสที่กำลังทำงานบนเซิร์ฟเวอร์ และอื่นๆ ซึ่งจะขึ้นอยู่กับคำสั่งที่ป้อนลงไป ผลลัพธ์การทำงานของคำสั่งเหล่านั้นจะแสดงค่าออกมาเป็นตัวเลข หรือเปอร์เซ็นต์ เพื่อให้ง่ายต่อความเข้าใจ

ก.1.1 คำสั่ง top

Top ถือว่าเป็นโปรแกรมหนึ่งที่ติดตั้งอยู่บนลีนุกซ์โดยอัตโนมัติ ใช้สำหรับการดูค่าการทำงานต่างๆของซีพียู และหน่วยความจำ รวมทั้งแสดงรายละเอียดโพรเซสที่กำลังทำงานอยู่บนเซิร์ฟเวอร์แบบเรียลไทม์ โดยปกติแล้วคำสั่ง top จะมีการทำงานใน 2 รูปแบบคือ แบบที่ติดต่อกับผู้ใช้งานโดยตรง โดยจะมีการรับค่าจากคีย์บอร์ดผู้ใช้งานและเกิดการแสดงผลทันที และรูปแบบที่ติดต่อกับผู้ใช้งานทางอ้อม โดยผู้ใช้งานจะต้องทำการป้อนคำสั่ง top และกำหนดรูปแบบการแสดงผลลัพธ์ก่อน จากนั้นโปรแกรม top จะทำงานตามที่ผู้ใช้กำหนดไว้

```
top - 12:49:35 up 55 days, 11:53,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  77 total,   1 running,  76 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3%us,  0.3%sy,  0.0%ni, 99.0%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:    250692k total,   228408k used,    22284k free,    71164k buffers
Swap:   240932k total,   102012k used,   138920k free,    61880k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2234 snmp      20   0  8844  944  664 S  0.3  0.4  33:29.48 snmpd
19754 root      20   0  8632 3028 2440 S  0.3  1.2   0:00.64 sshd
20958 root      20   0  2448 1140  904 R  0.3  0.5   0:00.46 top
    1 root      20   0  3084  292  240 S  0.0  0.1   0:04.92 init
    2 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      15  -5     0    0    0 S  0.0  0.0   1:05.57 ksoftirqd/0
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      15  -5     0    0    0 S  0.0  0.0   0:00.40 events/0
    7 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
    8 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 kstop/0
    9 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kintegrityd/0
   10 root      15  -5     0    0    0 S  0.0  0.0   0:08.01 kblockd/0
   11 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
   12 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kacpi_notify
   13 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 cqueue
   14 root      15  -5     0    0    0 S  0.0  0.0   0:00.01 ata/0
   15 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 ata_aux
   16 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 ksuspend_usbd
```

รูป ก.1 ผลลัพธ์จากการใช้คำสั่ง top แบบติดต่อกับผู้ใช้งานโดยตรง

ก.1.2 คำสั่ง ps

Ps เป็นคำสั่งที่ใช้ในการแสดงโพรเซสที่กำลังทำงานอยู่บนเครื่องเซิร์ฟเวอร์ แต่จะไม่มีโหมดติดต่อกับผู้ใช้งานโดยตรง ผู้ใช้งานจะต้องทำการกำหนดรูปแบบการแสดงผลก่อน คำสั่งนี้จึงจะแสดงผลลัพธ์ตามที่ผู้ใช้กำหนดไว้

```
root@monitor-project:/var/www/monitor/pages# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1   3084   292 ?        Ss    2010   0:04 /sbin/init
root         2  0.0  0.0      0     0 ?        S<    2010   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S<    2010   0:00 [migration/0]
root         4  0.0  0.0      0     0 ?        S<    2010   1:05 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<    2010   0:00 [watchdog/0]
root         6  0.0  0.0      0     0 ?        S<    2010   0:00 [events/0]
root         7  0.0  0.0      0     0 ?        S<    2010   0:00 [khelper]
root         8  0.0  0.0      0     0 ?        S<    2010   0:00 [kstop/0]
root         9  0.0  0.0      0     0 ?        S<    2010   0:00 [kintegrityd/0]
root        10  0.0  0.0      0     0 ?        S<    2010   0:08 [kblockd/0]
root        11  0.0  0.0      0     0 ?        S<    2010   0:00 [kacpid]
root        12  0.0  0.0      0     0 ?        S<    2010   0:00 [kacpi_notify]
root        13  0.0  0.0      0     0 ?        S<    2010   0:00 [cqueue]
root        14  0.0  0.0      0     0 ?        S<    2010   0:00 [ata/0]
root        15  0.0  0.0      0     0 ?        S<    2010   0:00 [ata_aux]
root        16  0.0  0.0      0     0 ?        S<    2010   0:00 [ksuspend_usbd]
root        17  0.0  0.0      0     0 ?        S<    2010   0:00 [khubd]
root        18  0.0  0.0      0     0 ?        S<    2010   0:00 [kseriod]
root        19  0.0  0.0      0     0 ?        S<    2010   0:00 [kmmcd]
root        20  0.0  0.0      0     0 ?        S<    2010   0:00 [btaddconn]
root        21  0.0  0.0      0     0 ?        S<    2010   0:00 [btdelconn]
root        22  0.0  0.0      0     0 ?        S     2010   0:01 [pdflush]
root        23  0.0  0.0      0     0 ?        S     2010   0:18 [pdflush]
root        24  0.0  0.0      0     0 ?        S<    2010   0:01 [kswapd0]
```

รูป ก.2 ผลลัพธ์จากการใช้คำสั่ง ps

ก.1.3 คำสั่ง sar

Sar เป็นโปรแกรมที่ใช้สำหรับการแสดงผลการทำงานของซีพียูเช่นเดียวกัน แต่เนื่องจากโปรแกรมนี้ไม่ได้ถูกติดตั้งมาตั้งแต่แรก จึงจำเป็นต้องมีการติดตั้งโดยผู้ใช้งานเอง ซึ่งการติดตั้งสามารถทำได้ด้วยการพิมพ์คำสั่ง apt-get install sysstat จากนั้นจึงสามารถใช้งานโปรแกรม sar ได้ตามปกติ โปรแกรม sar นี้จะมีการทำงานแบบติดต่อกับผู้ใช้ทางอ้อม กล่าวคือ จำเป็นต้องกำหนดรูปแบบการแสดงผลลัพธ์ก่อน แล้วจึงจะได้ผลลัพธ์ตามที่ผู้ใช้งานกำหนดไว้

```
\root@monitor-project:/var/www/monitor/pages# sar -u 1
Linux 2.6.28-11-server (monitor-project)        02/14/2011      _i686_  (1 CPU)

12:51:10 PM       CPU      %user      %nice    %system    %iowait     %steal      %idle
12:51:11 PM       all       1.00       0.00       0.00       0.00       0.00      99.00
12:51:12 PM       all       1.00       0.00       0.00       0.00       0.00      99.00
12:51:13 PM       all       1.00       0.00       5.00       0.00       0.00      94.00
12:51:14 PM       all       0.00       0.00       3.00       0.00       0.00      97.00
12:51:15 PM       all       0.00       0.00       1.98       0.00       0.00      98.02
12:51:16 PM       all       0.00       0.00       1.00       0.00       0.00      99.00
12:51:17 PM       all       2.00       0.00       6.00       0.00       0.00      92.00
12:51:18 PM       all       0.00       0.00       3.00       0.00       0.00      97.00
12:51:19 PM       all       0.00       0.00       1.00       0.00       0.00      99.00
12:51:20 PM       all       1.00       0.00       0.00       0.00       0.00      99.00
12:51:21 PM       all       2.00       0.00       7.00       0.00       0.00      91.00
12:51:22 PM       all       0.00       0.00       1.00       0.00       0.00      99.00
12:51:23 PM       all       1.00       0.00       0.00       0.00       0.00      99.00
12:51:24 PM       all       0.00       0.00       3.00       0.00       0.00      97.00
12:51:25 PM       all       2.00       0.00       6.00       1.00       0.00      91.00
12:51:26 PM       all       0.00       0.00       1.98       0.00       0.00      98.02
12:51:27 PM       all       0.00       0.00       1.00       0.00       0.00      99.00
12:51:28 PM       all       0.00       0.00       1.00       0.00       0.00      99.00
12:51:29 PM       all       0.00       0.00       5.00       0.00       0.00      95.00
12:51:30 PM       all       2.00       0.00       0.00       0.00       0.00      98.00
12:51:31 PM       all       0.00       0.00       1.00       0.00       0.00      99.00
12:51:32 PM       all       1.00       0.00       1.00       0.00       0.00      98.00
```

รูป ก.3 ผลลัพธ์จากการใช้คำสั่ง sar

ก.2 คำสั่งที่เกี่ยวข้องกับการทำงานของหน่วยความจำ

คำสั่งเหล่านี้จะใช้ในการแสดงค่าของหน่วยความจำที่กำลังถูกใช้งานอยู่จากเซิร์ฟเวอร์ ซึ่งโดยส่วนมากค่าของหน่วยความจำมักจะถูกแสดงควบคู่ไปกับค่าการทำงานของซีพียู แต่สำหรับการพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสนั้น ได้ใช้การคำนวณจากไฟล์ที่อยู่ภายใต้ไดเรกทอรี่สำหรับเก็บข้อมูลโพรเซสโดยตรง ทำให้สามารถได้ข้อมูลที่ตรงและแม่นยำกว่าการเรียกใช้โปรแกรมหรือคำสั่งอื่นๆ เพราะโปรแกรมหรือคำสั่งเหล่านั้น ต่างก็ต้องมีการเรียกไฟล์นี้ขึ้นมาทำการคำนวณก่อนเหมือนกัน

```
root@monitor-project:/var/www/monitor/pages# cat /proc/meminfo
MemTotal:         250692 kB
MemFree:           24172 kB
Buffers:           70196 kB
Cached:            61288 kB
SwapCached:         2956 kB
Active:           120840 kB
Inactive:          86444 kB
Active(anon):      30348 kB
Inactive(anon):    45532 kB
Active(file):      90492 kB
Inactive(file):    40912 kB
Unevictable:           0 kB
Mlocked:               0 kB
HighTotal:             0 kB
HighFree:              0 kB
LowTotal:         250692 kB
LowFree:           24172 kB
SwapTotal:        240932 kB
SwapFree:         138848 kB
Dirty:                52 kB
Writeback:             0 kB
AnonPages:         74072 kB
Mapped:            15308 kB
Slab:              11752 kB
SReclaimable:       7560 kB
```

รูป ก.4 ผลลัพธ์จากการเรียกดูไฟล์ /proc/meminfo

ก.3 คำสั่งที่เกี่ยวข้องกับการทำงานของฮาร์ดดิสก์

การแสดงผลของคำสั่งที่เกี่ยวข้องกับการทำงานของฮาร์ดดิสก์จะมีรูปแบบทั้งเป็นเปอร์เซ็นต์และค่าในหน่วยต่างๆ โดยผู้ใช้งานสามารถกำหนดหน่วยที่ใช้แสดงผลได้ ซึ่งคำสั่งที่นิยมใช้ในการแสดงผลการทำงานของฮาร์ดดิสก์มี 2 คำสั่งหลักๆ คือ

ก.3.1 คำสั่ง df

ใช้ในการแสดงปริมาณฮาร์ดดิสก์ที่เหลืออยู่ในเซิร์ฟเวอร์ โดยในคอลัมน์แรกจะแสดงชื่อของอุปกรณ์ที่ถูกเชื่อมต่อกับเซิร์ฟเวอร์ซึ่งอยู่ภายใต้ไดเรคทอรี่ /dev คอลัมน์ถัดมาแสดงพื้นที่ทั้งหมดของอุปกรณ์เชื่อมต่อชิ้นนั้น ในคอลัมน์ที่สามจะแสดงปริมาณพื้นที่ที่ถูกใช้ไป คอลัมน์ที่สี่แสดงปริมาณพื้นที่ที่เหลืออยู่ในอุปกรณ์เชื่อมต่อ ถัดมาเป็นปริมาณที่ถูกใช้ไปในค่าของเปอร์เซ็นต์ และคอลัมน์สุดท้ายจะแสดงพื้นที่ที่ถูกใช้งานโดยอุปกรณ์เชื่อมต่อชิ้นนั้น

ในการใช้งานคำสั่ง df นั้นจำเป็นต้องกำหนดรูปแบบการแสดงผลก่อน แล้วผลลัพธ์จึงจะแสดงตามรูปแบบที่ได้กำหนดไว้

```
root@monitor-project:/var/www/monitor/pages# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda1              3889892   1286272   2406024  35% /
tmpfs                   125344         0    125344   0% /lib/init/rw
varrun                  125344        84    125260   1% /var/run
varlock                 125344         0    125344   0% /var/lock
udev                    125344       140    125204   1% /dev
tmpfs                   125344         0    125344   0% /dev/shm
lrm                     125344      2392    122952   2% /lib/modules/2.6.28-11-se
rver/volatile
```

รูป ก.5 ผลลัพธ์จากการเรียกใช้คำสั่ง df

ก.3.2 คำสั่ง du

คำสั่ง du นั้นจะเน้นไปในเรื่องของการดูปริมาณการใช้พื้นที่ฮาร์ดดิสก์ของแต่ละไฟล์ และสามารถใช้ในการดูปริมาณการใช้พื้นที่ฮาร์ดดิสก์ของแต่ละไดเรกทอรี่ได้อีกด้วย

```
root@monitor-project:/var/www/monitor/pages# du -b /var/www/monitor/
383078  /var/www/monitor/js
5248    /var/www/monitor/tmpl
26341   /var/www/monitor/pages
6678    /var/www/monitor/libs
168628  /var/www/monitor/rrd_/imgs
1221360 /var/www/monitor/rrd_
4414    /var/www/monitor/nbproject/private
8967    /var/www/monitor/nbproject
30983   /var/www/monitor/css/ui-lightness/images
69211   /var/www/monitor/css/ui-lightness
51063   /var/www/monitor/css/imgs
6064    /var/www/monitor/css/blue
135715  /var/www/monitor/css
1794843 /var/www/monitor/
```

รูป ก.6 ผลลัพธ์จากการเรียกใช้คำสั่ง du

ก.4 คำสั่งที่เกี่ยวข้องกับการทำงานของเว็บเซิร์ฟเวอร์

ในการพัฒนาระบบเฝ้าระวัง เตือนภัย และอพยพโพรเซสนั้น ได้เน้นไปทางการเฝ้าระวังระบบเว็บเซิร์ฟเวอร์ซึ่งจะใช้ Apache เป็นหลัก ทำให้ต้องมีคำสั่งในการแสดงผลลัพธ์สถานะของเว็บเซิร์ฟเวอร์ Apache เข้ามาเกี่ยวข้อง

ก.4.1 คำสั่ง apache2ctl status

เป็นคำสั่งที่ใช้ในการแสดงสถานะของเว็บเซิร์ฟเวอร์ Apache ซึ่งในเบื้องต้นไม่จำเป็นต้องมีการแก้ไขค่าต่างๆเพื่อให้มีการใช้งานได้ เพราะหลังจากติดตั้งเว็บเซิร์ฟเวอร์แล้วจะมีโมดูลสำหรับการแสดงค่าสถานะอย่างง่ายมาให้ แต่ถ้าหากต้องดูข้อมูลให้ละเอียดขึ้นก็จำเป็นต้องมีการแก้ไขค่าบางอย่างเพื่อยอมให้เว็บเซิร์ฟเวอร์มีการเก็บค่าต่างๆมากขึ้น โดยคำสั่งนี้สามารถดูได้ผ่านทางเว็บบราวเซอร์ และผ่านทางเทอร์มินัล ทำให้สะดวกต่อผู้ใช้งานมากขึ้น

```
root@monitor-project:/var/www/monitor/pages# apache2ctl status
Apache Server Status for localhost

Server Version: Apache/2.2.11 (Ubuntu) PHP/5.2.6-3ubuntu4.5 with Suhosin-Patch
Server Built: Mar 9 2010 21:04:15

ââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââ

Current Time: Monday, 14-Feb-2011 12:56:18 ICT
Restart Time: Friday, 11-Feb-2011 00:35:22 ICT
Parent Server Generation: 1
Server uptime: 3 days 12 hours 20 minutes 55 seconds
Total accesses: 249398 - Total Traffic: 297.7 MB
CPU Usage: u17.35 s91.18 cu0 cs0 - .0357% CPU load
.821 requests/sec - 1027 B/second - 1251 B/request
1 requests currently being processed, 6 idle workers

_____W_.........................................................
................................................................
................................................................
................................................................

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
```

รูป ก.7 ผลลัพธ์จากการเรียกใช้คำสั่ง apache2ctl status

ภาคผนวก ข

การติดตั้งเพิร์ล และการติดต่อระหว่างเซิร์ฟเวอร์กับไคลเอนต์

ข.1 การติดตั้ง และไลบรารีของเพิร์ล

เพิร์ลเวอร์ชันที่ทำงานบนยูนิกซ์ หรือลีนุกซ์ สามารถดาวน์โหลดได้ที่ http://www.perl.com ซึ่งเพิร์ลเป็นโปรแกรมที่มีการใช้งานในวงการอินเตอร์เน็ตอย่างกว้างขวาง ทำให้มีการพัฒนาไลบรารี หรือไดรเวอร์ สำหรับเรียกใช้งานเกี่ยวกับการเขียนโปรแกรมสำหรับ TCP/IP เป็นจำนวนมาก ซึ่งจะมีนามสกุลเป็น .pm ซึ่งในการเขียนโปรแกรมสามารถเรียกใช้ไลบรารี หรือไดรเวอร์ นามสกุล .pm เพื่อเขียนเพิร์ลสคริปต์ได้

ข.2 การติดตั้ง และไลบรารีของเพิร์ลบนยูนิกซ์ หรือลีนุกซ์

เพิร์ลเป็นโปรแกรมที่ถูกติดตั้งมาแล้วกับ Unix Standard ทุกเครื่อง ซึ่งแต่ละเครื่องจะแตกต่างกันที่เวอร์ชัน และสถานที่ติดตั้งบนไดเรกทอรีของเครื่องเท่านั้น ดังนั้นจึงไม่จำเป็นต้องติดตั้งเพิร์ลบนเครื่องยูนิกซ์ หรือลีนุกซ์อีก แต่จะต้องเพิร์ลให้เจอว่าถูกติดตั้งไว้ ณ ที่ใด เพื่อจะได้ใช้เป็น Heading ในการเขียนโปรแกรมเพิร์ล โดยใช้คำสั่ง " whereis perl " หรือ " nd / -name perl " ค้นหา ซึ่งจะบอกตำแหน่งที่ตั้งของอินเตอร์พรีเตอร์ของเพิร์ล ในลีนุกซ์ที่อยู่ของเพิร์ลมีดังนี้

โปรแกรม ข.1 ตำแหน่งที่ตั้งของอินเตอร์พรีเตอร์ของเพิร์ล

```
root@project :# whereis perl
perl:
/usr/bin/perl
/etc/perl
/usr/lib/perl
/usr/share/perl
/usr/share/man/man1/perl.1.gz
```

ข.3 ทดสอบการเขียนเพิร์ล

รูปแบบการแสดงผลลัพธ์ของโปรแกรมเพิร์ลสคริปต์ จะแสดงผลแบบ Command Line ดังตัวอย่างโปรแกรม helloworld.pl จะแสดงอักขระ โดยใช้คำสั่ง print ดังนี้

โปรแกรม ข.2 ผลลัพธ์ที่ได้จากการใช้คำสั่ง print

```
root@project :# more helloword.pl
print "Hello World !!\n";
root@project :# perl helloword.pl
Hello World !!
```

ข.4 การเขียนแพ็คเกจ และการเรียกใช้ "require"

โดยทั่วไปในหนึ่งโปรแกรมจะมีอย่างน้อยหนึ่งแพ็คเกจ ซึ่งก็คือ "main package" หมายความว่า ถ้าในไฟล์ .pl ไม่มีการใช้ "package" statement จะถูกสมมติให้เป็น main package อัตโนมัติ อย่างเช่นตัวแปร $temp มีค่าเท่ากับ $main::temp เราสามารถสร้าง packages อื่นๆได้โดยใช้ "package" statement ดังนี้

```
1   #!/usr/bin/perl
2   use DBI;
3   package configDB;
4
5   $username = 'root';
6   $password = 'xxxxxx';
7   $database = 'monitor';
8   $hostname = 'localhost';
9
10
```

รูป ข.1 การเขียนแพ็คเกจ (configDB.pl)

การเรียกใช้ไฟล์แพ็คเกจจากโปรแกรมอื่น สามารถทำได้โดยใช้ "require" เพื่ออ้างถึงไฟล์ที่เป็นแพ็คเกจ ดังนี้

```
1   #! /usr/bin/perl
2   $| = 1; # enable autoflush
3   use strict;
4   use IO::Socket;
5   use CGI qw( :standard );
6   use IO::File;
7   use Switch;
8   use JSON::XS;
9   require "configDB.pl";
```

รูป ข.2 การเรียกใช้ "require"

จากนั้นใช้ "::" (double colon) เพื่ออ้างถึงตัวแปร หรือโปรแกรมย่อยที่อยู่ในไฟล์แพ็คเกจที่อ้างถึง ดังนี้

```
29  sub updateDB{
30      my ($user,$pass,$db,$host) = (  $configDB::username,
31                                      $configDB::password,
32                                      $configDB::database,
33                                      $configDB::hostname);
34      my $jdata = shift;
35      my $address = shift;
```

รูป ข.3 การอ้างถึงตัวแปรที่อยู่ในไฟล์แพ็คเกจ

ข.5 ตัวอย่างการใช้ regular expression

ในการกำหนดรูปแบบการค้นหาคำ (Expression) ของภาษา Perl นั้นมีลักษณะเช่นเดียวกับ Linux เพราะเนื่องจากภาษา Perl เป็นภาษาที่จัดการเกี่ยวกับระบบ Linux จึงทำให้การกำหนด expression มีลักษณะเหมือนกัน

โปรแกรม ข.3 การ match path ของ directory

```
if ($path =~ /\/usr\/local\/lib\/perl5/) { ... } การ match
path ของ directory ที่เราต้องการ
$match = $colors =~ / red /; # match ' red '
$match = $colors =~ /\bred\b/i; # match 'red', 'RED', 'rEd'
...
$match = $colors =~ /^red/; # match บรรทัดที่ขึ้นต้นด้วย 'red'
$match = $colors =~ /red$/; # match บรรทัดที่ลงท้ายด้วย 'red'
```

นอกจากนี้ในการทำ string processing เราจะใช้ภาษาอื่นเข้ามาช่วยด้วยเช่น grep, awk, sed, cut เป็นต้น

ข.5.1 คำสั่ง df | grep '\/$' | awk '{if($1~/sda/)$4=$5};END{print $4}' | sed s/%//

การใช้คำสั่ง df เพื่อดูค่า disk space ต่าง ๆ แต่ถ้าสมมติเราสนใจ Use% ของ disk ที่mounted on root ('/') ดังรูป

```
root@project-3:~# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/project--3-root
                       3616396   1136480   2296212  34% /
tmpfs                   125344         0    125344   0% /lib/init/rw
varrun                  125344        80    125264   1% /var/run
varlock                 125344         0    125344   0% /var/lock
udev                    125344       148    125196   1% /dev
tmpfs                   125344         0    125344   0% /dev/shm
lrm                     125344      2392    122952   2% /lib/modules/2.6.28-11-server/volatile
/dev/sda5               233335     13628    207259   7% /boot
root@project-3:~#
```

รูป ข.4 ผลลัพธ์การใช้คำสั่ง df

ข.5.2 คำสั่ง df | grep '\/$'

คำสั่งนี้ทำงานดังนี้คือ จะรับผลลัพธ์มากจากคำสั่ง df แล้วส่งข้อผลลัพธ์ผ่าน pipe ซึ่งส่งผลลัพธ์ต่อไปให้ grep ในคำสั่งนี้จะเป็นการกรองบรรทัดที่ลงท้ายด้วย '/' ด้วย expression '$' ส่วน '\' คือตัวที่ทำให้ '/' ไม่ถูกมองว่าเป็นอักขระพิเศษ ดังรูป

```
root@project-3:~# df | grep '\/$'
                       3616396   1136520   2296172  34% /
root@project-3:~#
```

รูป ข.5 ผลลัพธ์การใช้คำสั่ง df | grep '\/$'

ข.5.3 คำสั่ง df | grep '\/$' | awk '{if($1~/sda/)$4=$5};END{print $4}'

จะเห็นว่าหลังจากผลลัพธ์ที่แล้ว เมื่อส่งผลลัพธ์มาให้ awk และทำการ print column ที่เป็น 34% ออกมา นอกจากนี้ awk ยังสามารถเขียนเงื่อนไขสำหรับการ print ได้อีกด้วย เช่นในตัวอย่างนี้จะเช็คว่า coloumn ที่หนึ่งซึ่งเป็น 3616396 ไม่ match กับ expression ~/sda/ ดังนั้นจึง print column ที่ 4 ของผลลัพธ์ที่แล้วออกมา ดังรูป

```
root@project-3:~# df | grep '\/$' | awk '{if($1~/sda/)$4=$5};END{print $4}'
34%
root@project-3:~#
```

รูป ข.6 ผลลัพธ์การใช้คำสั่ง df | grep '\/$' | awk '{if($1~/sda/)$4=$5};END{print $4}'

ข.5.4 คำสั่ง df | grep '\/$' | awk '{if($1~/sda/)$4=$5};END{print $4}' | sed s/%//

จะเห็นว่าจากผลลัพธ์ 34% พอผ่านมาให้ sed แล้ว sed จะทำการตัดเครื่องหมาย % ออกด้วยการแทนที่ด้วยช่องว่าง 0 character ดังนี้

```
root@project-3:~# df | grep '\/$' | awk '{if($1~/sda/)$4=$5};END{print $4}' | sed s/%//
34
root@project-3:~#
```

รูป ข.7 ผลลัพธ์การใช้คำสั่ง df | grep '\/$' | awk '{if($1~/sda/)$4=$5};END{print $4}' | sed s/%//

โดยการใช้คำสั่งเหล่านี้เข้ามาช่วยในการเขียนภาษา perl เราจะใช้คำสั่งผ่าน “ `` ” (double back quote) ดังนี้ `command` เช่น `df` เป็นต้น

ข.6 การติดต่อระหว่างเซิร์ฟเวอร์กับไคลเอนต์

ข.6.1 โมดูลพื้นฐานสำหรับโปรแกรมระบบเครือข่าย

ซ็อกเก็ต คือการเชื่อมต่อกันของอุปกรณ์ในระดับล่าง ซึ่งมีหน้าที่ช่วยให้การติดต่อสื่อสารระหว่างอุปกรณ์ผ่านเครือข่าย TCP/IP ให้สามารถเกิดขึ้นได้ ในภาษาเพิร์ลจะมีโมดูลสำหรับการเขียนโปรแกรมทางด้านระบบเครือข่ายมาไว้ให้แล้ว โดยอยู่ในไดเรกทอรีที่มีชื่อว่า Net ภายใน ไดเรกทอรี Net ได้บรรจุไฟล์นามสกุลพีเอ็มที่เกี่ยวข้องกับอินเทอร์เน็ตเซอร์วิสต่างๆ อาทิ เช่น Ping.pm และ FTP.pm ไว้

ข.6.2 การเขียนโปรแกรมผ่านซ็อกเก็ต และโมดูลซ็อกเก็ต

การเขียนโปรแกรมเครือข่าย TCP/IP แบบพื้นฐาน ในภาษาเพิร์ลได้พัฒนาโมดูลชื่อ IO::Socket เพื่อใช้ในการเขียนโปรแกรม

ข.6.2.1 พอร์ต

ทีซีพีโปรโตคอล และยูดีพีโปรโตคอล จะใช้พอร์ตสำหรับการส่งข้อมูล ซึ่งเปรียบเสมือนท่อรับส่งข้อมูลระหว่างเซิร์ฟเวอร์กับไคลเอนต์ ซึ่งมีขนาด 16 บิต ทำให้คอมพิวเตอร์แต่ละตัวสามารถมีหมายเลขพอร์ตได้ตั้งแต่ 0 ถึง 65535 ซึ่งข้อมูลเหล่านี้ถูกเก็บไว้ที่ไฟล์ /etc/service

1) พอร์ตหมายเลข 1 ถึง 255 ถูกสงวนไว้สำหรับบริการมาตรฐาน เช่น พอร์ต 80 สำหรับ www

2) พอร์ตหมายเลข 1 ถึง 1023 ถูกสงวนไว้ให้สำหรับเซิร์ฟเวอร์ทำงานต่างๆ

3) พอร์ตหมายเลข 1024 ถึง 65535 เป็นหมายเลขพอร์ตที่ว่างซึ่งนำมาใช้งาน

ข.6.2.2 ซ็อกเก็ต

การรับส่งข้อมูลผ่านระบบเครือข่ายทีซีพีไอพี จะต้องใช้ซ็อกเก็ต โดยทั้งฝั่งเซิร์ฟเวอร์ และไคลเอนต์จะต้องสร้างซ็อกเก็ตขึ้นมาเพื่อเชื่อมต่อกัน ซึ่งเลือกได้ 2 แบบ ดังนี้

1) SOCK_STREAM ใช้กับทีซีพีโปรโตคอล ซ็อกเก็ตแบบนี้จะใช้สำหรับแอปพลิเคชั่นที่ต้องการมีความน่าเชื่อถือในการส่งข้อมูล จึงต้องมีการสร้างการ

เชื่อมต่อขึ้นมาก่อน อาทิ เช่น ก่อนส่งข้อมูลเซิร์ฟเวอร์กับไคลเอนต์จะต้องมีการสร้างการเชื่อมต่อขึ้นมาก่อน

2) SOCK_DGRAM ใช้กับยูดีพีโปรโตคอลซ็อกเก็ตแบบนี้จะใช้สำหรับแอปพลิเคชั่นที่ไม่ต้องการความน่าเชื่อถือในการส่งข้อมูลสูงมากนัก จึงไม่ต้องสร้างการเชื่อมต่อขึ้นมาก่อน อาทิ เช่น การส่งข้อมูลระหว่างเซิร์ฟเวอร์กับไคลเอนต์ไม่ต้องมีการสร้างการเชื่อมต่อขึ้นมาก่อน

ข.6.2.3 พารามิเตอร์ของซ็อกเก็ตสำหรับโมดูล IO::Socket

ตารางสรุปพารามิเตอร์ของซ็อกเก็ตสำหรับการเขียนโปรแกรมเครือข่ายทีซีพีไอพี โดยใช้โมดูล IO::Socket

ตาราง ข.1 พารามิเตอร์ต่างๆของซ็อกเก็ต

พารามิเตอร์	ความหมาย	รูปแบบการเขียน
PeerAddr	ไอพีแอดเดรส หรือชื่อของโฮสปลายทาง	<hostname>[:<port>]
PeerPort	หมายเลขพอร์ตปลายทางที่ซ็อกเก็ตต้องการจะติดต่อด้วย	<service>[<no.>] หรือ <no.>
LocalAddr	ไอพีแอดเดรส หรือชื่อของโฮส	<hostname>[:<port>]
LocalPort	หมายเลขพอร์ตต้นทางที่ซ็อกเก็ตกำลังทำงานอยู่	<service>[<no.>] หรือ <no.>
Proto	ชื่อของโปรโตคอล หรือหมายเลขของโปรโตคอลที่ซ็อกเก็ตใช้	tcp,udp
Type	ประเภทของซ็อกเก็ต	SOCK_STREAM, SOCK_DGRAM
Listen	จำนวนซ็อกเก็ตสูงสุดที่อนุญาตให้เก็บคิวไว้ได้ โดยใช้ได้เฉพาะที่ฝั่งเซิร์ฟเวอร์	ระบุเป็นตัวเลข
Reuse	Reuse => 1 เพื่อสั่งให้ตัวแปร SO_REUSEADDR ของซ็อกเก็ตมีค่าเป็น 1 ก่อนใช้คำสั่ง blind เพื่อป้องกันปัญหา Address already in use	ระบุเป็นตัวเลข 1
Timeout	ใช้ตั้งเวลาการรอคอยของซ็อกเก็ต	ระบุเป็นตัวเลข (หน่วยวินาที)

โปรแกม ข.4 การสร้างซ็อกเก็ตสำหรับฝั่งเซิร์ฟเวอร์

```
$sock = IO::Socket->new(Listen=>2,
LocalAddr => 'localhost',
LocalPort => 8899,
Proto     => 'tcp');
$sock = IO::Socket::INET->('127.0.0.1:25')
```

โปรแกม ข.5 การสร้างซ็อกเก็ตสำหรับฝั่งไคลเอนต์

```
$sock = IO::Socket->new(PeerAddr => 'www.perl.org',
PeerPort => 'http(80)',
Proto     => 'tcp');
$sock = IO::Socket::INET->new(PeerAddr =>
                    'localhost:smtp(25)');
$sock = IO::Socket::INET->new PeerPort => 8899,
                PeerAddr => inet_ntoa(INADDR_BROADCAST),
                Proto=> udp,
                LocalAddr => 'localhost'
                Broadcast => 1)
                Or die "Can't blind : $@\n";
```

ข.6.2.4 เมธอดของโมดูลซ็อกเก็ต

ตารางสรุปเมธอดของซ็อกเก็ตสำหรับการเขียนโปรแกรมเครือข่าย TCP/IP โดยใช้โมดูล IO::Socket

ตาราง ข.2 เมธอดต่างๆของโมดูลซ็อกเก็ต

เมธอด	ความหมาย	รูปแบบการเขียน
Accept	ซ็อกเก็ตที่ฝั่งเซิร์ฟเวอร์ยอมรับการเชื่อมต่อจากซ็อกเก็ตของฝั่งไคลเอนต์ที่ฝั่งเซิร์ฟเวอร์เท่านั้น	$socket_2 = $my_socket->accept;
Close	ปิดการเชื่อมต่อ	Close ($my_socket)
Recv	รับข้อมูลจากซ็อกเก็ต	$my_socket-> recv($text,128);
Send	ส่งข้อมูล (เขียนข้อมูล) ไปยังซ็อกเก็ต	$my_socket- >send($msg);
Peeraddr	คืนค่าหมายเลขซ็อกเก็ตของเครื่องคอมพิวเตอร์ที่กำลังทำการเชื่อมต่อมา (Remote Connection)	$client_sock = $socket->peeraddr

ตาราง ข.2 เมธอดต่างๆของโมดูลซ็อกเก็ต (ต่อ)

เมธอด	ความหมาย	รูปแบบการเขียน
peerhost	คืนค่าหมายเลขไอพีของเครื่องคอมพิวเตอร์ที่กำลังทำการเชื่อมต่อมา (Remote Connection)	$client_add = $socket->peerhost
Peerport	คืนค่าหมายเลขพอร์ตของเครื่องคอมพิวเตอร์ที่กำลังทำการเชื่อมต่อมา(Remote Connection)	$client_port = $socket->peerport
Sockaddr	คืนค่าหมายเลขซ็อกเก็ตของเครื่องคอมพิวเตอร์ที่กำลังดำเนินการอยู่	$server_add = $socket->sockaddr
Sockhost	คืนค่าหมายเลขไอพีของเครื่องคอมพิวเตอร์ที่กำลังดำเนินการอยู่	$server_host = $socket->sockhost
Sockport	คืนค่าหมายเลขพอร์ตของเครื่องคอมพิวเตอร์ที่กำลังดำเนินการอยู่	$server_port = $socket->sockport

ข.6.2.5 การสร้างซ็อกเก็ต

การสร้างซ็อกเก็ต มี 2 แบบ คือสำหรับฝั่งเซิร์ฟเวอร์ และฝั่งไคลเอนต์

โปรแกรม ข.6 การสร้างซ็อกเก็ตสำหรับฝั่งเซิร์ฟเวอร์

```
use IO::Socket::INET;

# Create a new Socket
$my__socket=new IO::Socket::INET->new(LocalPort=>9999,
Proto=>'udp');
$date1 = `date`;
print "Socket Server Program "Start on $date1 \n";

#Keep Receive MSG from Client
print "\nReceiving Msg from client\n";
$i=0;

while(1){
        ($sec,$min,$hour)=localtime time;
        $i = $i+1;
        $my__socket->recv($text,65535);
        $client_addr = $my__socket->peerhost;
        print "Client Connected From : $client_addr";
        if($text ne 'End'){
                print "\nReceived msg
[$i][$hour:$min:$sec]>>\n",$text,"\n";
        }

         else{
                $date2 =`date`;
                print "Client is terminated on $date2"; exit
1;
              }
}
```

โปรแกรม ข.7 การสร้างซ็อกเก็ตสำหรับฝั่งไคลเอนต์

```
$remote_host = '161.246.34.191';
$remote_port = '9999';

use IO::Socket::INET;
$my__socket = new IO::Socket::INET-
>new(PeerPort=>$remote_port,
                                        Proto=>'udp',

PeerAddr=>$remote_host)
                    or die "Can't Connect to
          $remote_host:$remote_port : $!\n";

$date = `date`;
$command = `top -b -n1 | awk 'BEGIN{OFS="\t"; print
"PID\tUSER\tSTAT\tCPU\tMEM\tTIME\tPROC"}; NR>7{print
\$1,\$2,\$8,\$9,\$10,\$11,\$12}' | sort -n`;

print "Socket-Client Program : Start on $date";

#Send Command
       print "Sending Command\n";

       if($my__socket->send($command)){
             print "...done command1\n";
       }
       else{
             print "Failed to Send\n";
       }
```

ข.6.2.6 การส่ง (เขียน) ข้อมูลผ่านซ็อกเก็ต

เมื่อการเชื่อมต่อเสร็จแล้ว เราจะสามารถรับส่งข้อมูลกันได้ โดยการส่งข้อมูลไปยังซ็อกเก็ตนั้น สามารถทำได้โดยใช้เมธอด send() ในการรับส่งข้อมูล สำหรับการใช้ฟังก์ชัน send() มีวิธีดังนี้ $socket->send("Hello World") ถ้าต้องการตรวจสอบว่าสามารถส่งได้หรือไม่ สามารถเขียนได้ดังนี้ถ้าหากการส่งข้อมูลไม่สำเร็จ โปรแกรมจะจบการทำงาน และจะแสดงผลข้อผิดพลาดที่ถูกเก็บไว้ในตัวแปร $!

โปรแกรม ข.8 ตรวจสอบว่าส่งข้อมูลได้หรือไม่

```
$msg1 = "Hello, I will send message";
$socket->send($msg1) or die "send problem: $!";
```

ข.6.2.7 การรับ (อ่าน) ข้อมูลผ่านซ็อกเก็ต

สำหรับการรับ หรือการอ่านข้อมูลจากซ็อกเก็ต สามารถทำได้โดยใช้เมธอด recv() ซึ่งมีคำสั่งดังนี้ $socket->recv($msg.buffer); โดยข้อมูลจะถูกเก็บไว้ในตัวแปร $msg ส่วน buffer คือความยาวของข้อความมากที่สุดที่เป็นไปได้

ภาคผนวก ค

รูปแบบคำสั่งที่ใช้ใน Round Robin Database Tool

ค.1 โครงสร้าง Round Robin Database

ค.1.1 การสร้าง Round Robin Database

รูปแบบคำสั่ง Create ใน RRDTool เป็นการสร้างไฟล์ Round Robin Database ขึ้นมา โดยไฟล์ที่ถูกสร้าง ประกอบด้วยขนาดของข้อมูลที่ต้องการเก็บทั้งหมด และในแต่ละฟิลด์ของข้อมูลจะเป็นประเภท Unknow เนื่องจากยังไม่มีข้อมูลเก็บอยู่

โปรแกรม ค.1 คำสั่งสร้างไฟล์ Round Robin Database

```
rrdtool create filename [--start|-b start time] [--
step|-s step] [--no-overwrite] [DS:ds-name:DST:dst
arguments] [RRA:CF:cf arguments]
```

1) Filename ชื่อของไฟล์ RRD ที่ต้องการสร้าง ไฟล์ RRD เป็นนามสกุล .rrd
2) --start|-b start time กำหนดเวลาเป็นหน่วยวินาที เมื่อค่าข้อมูลแรกสุดถูกเพิ่มลงในฐานข้อมูล RRD โดย RRDTool จะไม่ทำการตอบรับข้อมูลอื่นๆจนกว่าจะหมดเวลาที่กำหนดไว้
3) --step|-s step กำหนดช่วงเวลาพื้นฐาน (Base Interval) ในหน่วยวินาที เป็นการกำหนดช่วงเวลาที่ข้อมูลถูกป้อนเข้าไปในฐานข้อมูล RRD
4) --no-overwrite กำหนดไม่ให้มีไฟล์ RRD ชื่อเดียวกัน
5) DS:ds-name:DST:dst arguments โดยปกติ Data Source (DS) ของฐานข้อมูล RRD เดียว สามารถรับข้อมูลจากหลายๆแหล่งข้อมูล (Data Source) ได้ โดย ds-name เป็นชื่อเพื่อใช้ในการอ้างอิงแหล่งข้อมูลต่างๆจากฐานข้อมูล RRD โดย ds-name ประกอบด้วยอักขระ ความยาวตั้งแต่ 1 ถึง 19 ตัว ส่วน Data Source Type (DST) ค่าอาถิวเมนต์ของแหล่งข้อมูลจะขึ้นอยู่กับการกำหนดประเภทของ DST โดยประเภทของ DST ประกอบด้วย GAUGE COUNTER DERIVE ABSOLUTE และ COMPUTE เป็นต้น

โปรแกรม ค.2 คำสั่ง Data Source Type

```
DS:ds-name:GAUGE | COUNTER | DERIVE |
ABSOLUTE:heartbeat:min:max
```

ค.1.2 RRA:CF:cf arguments

RRD จะเก็บข้อมูลไว้ใน Round Robin Archive (RRA) โดย RRA สามารถเก็บจำนวนของค่าข้อมูล หรือเก็บค่าทางสถิติของแต่ละแหล่งข้อมูลที่กำหนดได้ เมื่อข้อมูลถูกป้อนลงใน RRD ข้อมูลจะมีขนาดพอดีกับช่วงเวลาของความยาวที่กำหนดไว้ แล้วเรียกข้อมูลเหล่านี้ว่า Primary Data Point (PDP)

ข้อมูลจะถูกจัดเก็บด้วยฟังค์ชันรวบรวมข้อมูล (Consolidation Function) ซึ่งสามารถมีการรวบรวมข้อมูลได้หลายๆฟังค์ชัน โดยสามารถทำการรวมข้อมูล PDP ผ่านฟังค์ชันรวบรวมข้อมูลแบบต่างๆ เช่น AVERAGE MIN MAX LAST เป็นต้น

โดย AVERAGE เก็บข้อมูลค่าเฉลี่ยของ PDP ไว้ ส่วน MIN เก็บข้อมูลค่าที่น้อยที่สุดของ PDP ต่อไป MAX เก็บข้อมูลค่าที่มากที่สุดของ PDP และ LAST เก็บข้อมูลล่าสุดของ PDP ไว้

โปรแกรม ค.3 คำสั่ง RRA

```
RRA:AVERAGE | MIN | MAX | LAST:xff:steps:rows
```

1) XFile Factor (XFF) เป็นการกำหนดช่วงเวลาการรวมข้อมูลจากข้อมูลที่ไม่ทราบประเภท ในขณะที่การรวมข้อมูลเป็นข้อมูลที่ทราบประเภท
2) Steps กำหนดจำนวน PDP ซึ่งจะถูกใช้ในการสร้างการรวมข้อมูลเพื่อเข้าสู่การจัดเก็บข้อมูลต่อไป
3) Rows กำหนดจำนวนข้อมูล ซึ่งจะถูกเก็บไว้ใน RRA โดยข้อมูลต้องมากกว่าศูนย์

ค.2 การสร้างกราฟใน Round Robin Database Tool

ฟังค์ชันกราฟของ RRDTool ถูกใช้เพื่อนำเสนอข้อมูลจากฐานข้อมูล RRD โดยแสดงข้อมูลเป็นภาพ และแสดงรายงานเชิงตัวเลข โดยการใช้คำสั่ง Fetch เป็นการดึงค่าจากฐานข้อมูล แล้วนำมาพล็อตกราฟตามค่าที่ดึงมาได้

โปรแกรม ค.4 คำสั่งสร้างกราฟ

```
rrdtool graph|graphv filename [option ...] [data
definition ...] [data calculation ...] [variable
definition ...] [graph element ...] [print element ...]
```

ค.2.1 Option

1) Filename ชื่อกราฟที่ต้องการสร้าง และจะต้องเป็นไฟล์นามสุกล .png .svg หรือ .eps เท่านั้น

2) Time Range กำหนดอนุกรมเวลาตั้งแต่จุดเริ่มต้นถึงจุดสิ้นสุดที่ต้องการแสดงออกมาด้วยข้อมูลที่อยู่ใน RRA

โปรแกรม ค.5 คำสั่ง Time Range

```
[-s|--start time] [-e|--end time] [-S|--step seconds]
```

3) Label กำหนดข้อความที่ด้านบนของกราฟในแนวนอน และกำหนดข้อความในแนวตั้งด้านซ้ายของกราฟ

โปรแกรม ค.6 คำสั่ง Label

```
[-t|--title string] [-v|--vertical-label string]
```

4) Size โดยปกติ ความว้าง และความสูงของภาพกำหนดไว้ที่ 400 พิกเซล และ 100 พิกเซล ถ้าระบุออฟชั่น --full-size-mode จะทำให้ขนาดความกว้าง และความสูงของรูปเอาท์พุท และภาพที่ออกมาจะมีการปรับขนาดให้พอดีโดยอัตโนมัติ ถ้าระบุออฟชั่น only-graph และตั้งค่าความสูงน้อยกว่า 32 พิกเซล จะได้ภาพกราฟเล็กๆ เพื่อใช้เป็นไอคอน

โปรแกรม ค.7 คำสั่ง Size

```
[-w|--width pixels][-h|--height pixels][-j|--only-
graph][-D|--full-size-mode]
```

5) Limits โดยปกติค่าเริ่มต้นของกราฟจะทำการปรับค่าสเกล(Scale) ให้อัตโนมัติ เพื่อที่จะปรับค่าแกน Y ตามช่วงของข้อมูล แต่สามารถปรับเปลี่ยนการตั้งค่านี้ใหม่ได้ โดยตั้งค่าเป็น limits จะทำให้ค่าแกน Y แสดงค่าช่วงที่น้อยสุดจาก limits ต่ำสุดถึง limits สูงสุด

โปรแกรม ค.8 คำสั่ง Limits

```
[-u|--upper-limit value][-l|--lower-limit value][-r|--
rigid]
```

6) X-Axis สามารถกำหนดค่ากราฟในแนวแกน X ได้โดยการตั้งค่าอัตโนมัติ หรือกำหนด ค่าเองก็ได้ โดยตารางสามารถระบุเลือกแสดงเป็นวินาที นาที ชั่วโมง วัน สัปดาห์ เดือน และปีได้ตามต้องการ

โปรแกรม ค.9 คำสั่ง X-Axis

```
[-x|--x-grid GTM:GST:MTM:MST:LTM:LST:LPR:LFM]
```

7) Y-Axis เส้นกราฟแกน Y จะแสดงให้เห็นช่วงเวลาในแต่ละสเต็ป และป้ายชื่อจะถูกวางไว้ทุกๆเส้นของกราฟ โดยค่าเริ่มต้นกำหนดให้อยู่ในโหมดอัตโนมัติ

โปรแกรม ค.10 คำสั่ง Y-Axis

```
[-y|--y-grid grid step:label factor]
```

8) Right Y Axis แกนที่สองจะถูกดึงไปทางขวาของกราฟ โดยจะเชื่อมโยงกับแกนด้านซ้ายผ่านสเกลข้อมูล และเปลี่ยนค่าพารามิเตอร์ นอกจากนี้ยังสามารถกำหนดป้ายชื่อสำหรับกแกนด้านขวาได้อีกด้วย

โปรแกรม ค.11 คำสั่ง Right Y Axis

```
[--right-axis scale:shift] [--right-axis-label label]
```

9) Legend เพื่อวางคำอธิบายที่ด้านข้างของกราฟที่กำหนด โดยเริ่มต้นจะอยู่ทางทิศใต้ ในตำแหน่งทางทิศตะวันตกหรือทิศตะวันออกจะต้องเพิ่มบรรทัดใหม่ด้วยตนเอง

โปรแกรม ค.12 คำสั่ง Legend

```
[--legend-position=(north|south|west|east)]
```

10) Miscellaneous ปรับแต่งแบบอักษรที่จะใช้เป็บข้อความต่างๆบนกราฟ RRD ซึ่งประกอบด้วย DEFAULT TITLE AXIS UNIT LEGEND WATERMARK เป็นต้น

โปรแกรม ค.13 คำสั่ง Miscellaneous

```
[-n|--font FONTTAG:size:[font]]
```

ค.2.2 Data and Variable

ทั้งสามคำสั่งนี้ประกอบด้วย เป็นคำสั่งเพื่อดึงข้อมูลจากไฟล์ RRD มาแสดงผลเป็นกราฟ โดยกำหนดตัวแปร (vname) ซึ่งประกอบด้วยอักขระ A-Z, a-z, 0-9, -, _ ความยาวไม่เกิน 255 ตัวอักษร

คำสั่ง DEF จะเป็นการเรียกข้อมูลจากไฟล์ RRD โดยกำหนดตัวแปร มีค่าเริ่มต้น RRA ซึ่งประกอบไปด้วยข้อมูลที่รวบรวมไว้อย่างถูกต้อง สามารถเพิ่มความละเอียดในออฟชั่น -- step เข้าไปและเพิ่ม --start --end เพื่อกำหนดช่วงเวลาของข้อมูลให้เป็น เช่นเดียวกับกราฟ

โปรแกรม ค. 14 คำสั่ง DEF

```
DEF:<vname>=<rrdfile>:<ds-
name>:<CF>[:step=<step>][:start=<time>][:end=<time>][:r
educe=<CF>]
```

ค.2.3 GRAPH

GPRINT เป็นการแสดงภาพ และรายงานที่ต้องการ โดยสามารถแสดงค่าพร้อมเวลาบนกราฟ

โปรแกรม ค.15 คำสั่ง GPRINT

```
GPRINT:vname:CF:format
```

ค.2.4 Escaping the colon

เครื่องหมายโคลอน(:) ในอากิวเมนต์ Legend เพื่อบ่งบอกจุดสิ้นสุดของ Legend นั้นๆ โดยการป้อนเครื่องหมายโคลอนจะต้องปิดด้วยเครื่องหมายแบคสแลช (\)

ค.2.5 Strings Formatting

เพื่อจัดรูปแบบตัวอักษรที่อยู่ด้านล่างของกราฟ โดยพิมพ์อักขระต่อท้ายอักขระพิเศษแบคสแลชที่ท้ายข้อความ โดยในแต่ละอักขระมีความหมายดังนี้

\j เพื่อจัดให้พอดี

\l เพื่อจัดชิดซ้าย

\r เพื่อจัดชิดขวา

\c เพื่อจัดให้อยู่ตำแหน่งตรงกลาง

โปรแกรม ค.16 คำสั่ง String Formatting

```
GPRINT:a:MAX:%lf%s \j
```