โปรแกรมมอนิเตอร์ระบบประมวลผลแบบกลุ่มลินุกซ์

LINUX CLUSTER COMPUTING MONITORING PROGRAM

นายคมกริช เย็นศิริกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

โปรแกรมมอนิเตอร์ระบบประมวลผลแบบกลุ่มลินุกซ์
LINUX CLUSTER COMPUTING MONITORING PROGRAM

โดย

นายคมกริช เย็นศิริกุล

อาจารย์ที่ปรึกษา

ผศ. บรรจง ปิยธำรง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

ปริญญานิพนธ์ปีการศึกษา 2543

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมมอนิเตอร์ระบบประมวลผลแบบกลุ่มลินุกซ์

LINUX CLUSTER COMPUTING MONITORING PROGRAM

ผู้จัดทำ

1. นาย คมกริช เย็นศิริกุล รหัสประจำตัว 40010100

_____ อาจารย์ที่ปรึกษา

(ผศ. บรรจง ปิยธำรง)

โปรแกรมมอนิเตอร์ระบบประมวลผลแบบกลุ่มลินุกซ์

นายคมกริช เย็นศิริกุล 40010100
ผศ. บรรจง ปิยธำรง อาจารย์ที่ปรึกษา
ปีการศึกษา 2543

บทคัดย่อ

ในงานบางอย่างเช่น การพยากรณ์อากาศ มีการประมวลผลที่ละเอียดซับซ้อนและต้องใช้ข้อมูลที่เกี่ยวข้องมาก ,งานบางอย่างเช่นการประมวลภาพสามมิติ อาจต้องใช้เวลาในการประมวลผลหลายสัปดาห์ ซึ่งการประมวลผลงานประเภทนี้จำเป็นต้องใช้ระบบขนาดใหญ่เช่น ซุปเปอร์คอมพิวเตอร์ ในการประมวลผล เพื่อประหยัดต้นทุนทางด้านเวลาให้มากที่สุด ซึ่งซุปเปอร์คอมพิวเตอร์เหล่านี้จะจำกัดการใช้งานเฉพาะกลุ่มคนบางกลุ่มเท่านั้นเนื่องจากมีต้นทุนด้านราคาที่สูง

ในปัจจุบันประสิทธิภาพของเครื่องคอมพิวเตอร์ส่วนบุคคลหรือพีซี มีความสามารถเพิ่มขึ้นแบบก้าวกระโดด คือเพิ่มขึ้นแบบรวดเร็วมากเมื่อเทียบกับระบบใหญ่ๆอย่างเมนเฟรม จึงมีการนำเอาเครื่องพีซีหลายๆเครื่องมาต่อเข้าด้วยกันผ่านระบบเครือข่ายโดยมองกลุ่มก้อนของพีซีเหล่านั้นเป็นระบบหนึ่งเดียวเพื่อเพิ่มความสามารถในการประมวลผลให้เทียบเท่าหรือใกล้เคียงเครื่องซุปเปอร์คอมพิวเตอร์ ในขณะที่ราคาโดยรวมของระบบต่ำกว่า และเราเรียกระบบนี้ว่า ระบบประมวลผลแบบกลุ่มหรือคลัสเตอร์

ระบบประมวลผลแบบกลุ่มนั้นอาจจะประกอบไปด้วยเครื่องพีซีหลายๆเครื่อง ในบางระบบอาจมีมากถึงหลายร้อยเครื่อง ซึ่งทำให้การจัดการและเฝ้าติดตามการทำงานของทั้งระบบเป็นไปได้ยาก วิทยานิพนธ์ฉบับนี้จึงเป็นการสร้างระบบประมวลผลตัวอย่างขึ้นมาทดสอบการทำงาน และสร้างโปรแกรมเฝ้าติดตามการทำงานของระบบตัวอย่างที่เราสร้างขึ้นคลัสเตอร์ เพื่อเพิ่มความสดวกให้กับผู้ใช้งานและผู้ดูแลระบบให้มากที่สุด

LINUX CLUSTER COMPUTING MONITORING PROGRAM

Komkrit Yensirikul

Assist. Prof. Bunjong Piyathumrong

Advisor

ABSTRACT

Some application such as weather prediction , have a complexity in computation and use large scale of data to compute ,in 3D graphic rendering may give output after several weeks. So that task require supercomputer computation ,because it can reduce more cost of time , in spite of supercomputer limit in few group of people and application cause of it's extremely price.

Recently, PC quickly increase performance compare to mainframe or supercomputer , so there is way to group several PC via network and view group of that PC in single system image ,call cluster to increase performance and can comparable with supercomputer but more less price

In this thesis cover building sample cluster and cluster monitoring program , Use be tool for user and administrator in cluster system

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี ถ้าไม่ได้รับความช่วยเหลือจากบุคคลต่อไปนี้ อาจารย์บรรจง ปิยธำรง อาจารย์ที่ปรึกษาวิทยานิพนธ์ที่ได้แนะแนวทางและได้ให้คำปรึกษารวมถึงแหล่งข้อมูลบางส่วนของโครงงานนี้ด้วย ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

และต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ เอาใจใส่เสมอมาในทุก ๆ ด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

คมกริช เย็นศิริกุล

สารบัญ

สารบัญตาราง

สารบัญรูปภาพ

หน้าที่

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันซอฟต์แวร์ต่างๆได้มีการพัฒนาไปอย่างมาก การพัฒนาทางด้านคอมพิวเตอร์ทำให้เกิดอัลกอริทึมหรือกระบวนการทางความคิดต่างๆขึ้นมากมาย ทำให้ซอฟต์แวร์เหล่านั้นมีความสามารถเพิ่มขึ้นมาก อีกทั้งการพัฒนาทางด้านวิทยาศาสตร์และเทคโนโลยีในปัจจุบันก้าวหน้าไปมากและได้มีการนำเอาคอมพิวเตอร์มาใช้ในการพัฒนาและประมวลมากขึ้น ข้อดีของการนำเอาคอมพิวเตอร์มาใช้ก็คือ คอมพิวเตอร์มีการประมวลผลที่เป็นแบบอัตโนมัติ สดวกและรวดเร็ว คอมพิวเตอร์ใช้เวลาในการประมวลผลน้อยกว่าเมื่อเทียบกับคน คอมพิวเตอร์มีความถูกต้องแม่นยำ อีกทั้งคอมพิวเตอร์ยังมีเสถียรภาพในการทำงาน สามารถทำงานได้ 24 ชั่วโมงโดยไม่ต้องมีการหยุดพักเหมือนคน และสุดท้ายคอมพิวเตอร์สามารถบันทึกข้อมูลได้และไม่สูญหาย ซึ่งคนอาจมีขีดจำกัดของความจำ อาจจะจำได้ไม่มากและจำได้ในระยะเวลาสั้นๆ อีกทั้งคอมพิวเตอร์ยังเหมาะกับการประมวลผลปริมาณมากๆและงานซ้ำๆ มากกว่าคนอีกด้วย ดังนั้นการนำเอาคอมพิวเตอร์มาใช้ทำให้การพัฒนางานทางด้านวิทยาศาสตร์และเทคโนโลยีเป็นไปได้อย่างง่ายดาย ในปัจจุบันได้มีการคิดทฤษฎีต่างๆขึ้นมากมายทำให้การคำนวณงานทางด้านวิทยาศาสตร์ใช้กระบวนวิธีที่ซับซ้อนขึ้นเพื่อให้ได้ผลลัพธ์ตามที่ต้องการ ซึ่งกระบวนวิธีบางอย่างก็ต้องใช้การประมวลผลปริมารสูงมาก (mass intensive computation) ดังนั้นความจำเป็นของการใช้ฮาร์ดแวร์ที่มีประสิทธิภาพสูงจึงเพิ่มตามความสามารถของซอฟต์แวร์ไปด้วย ยกตัวอย่างเช่นการทำนายสภาพอากาศเป็นงานที่ต้องใช้ข้อมูลปริมาณมากในการคำนวณ อีกทั้งยังมีอัลกอริทึมที่ซับซ้อน การประมวลผลจำเป็นต้องใช้ซุปเปอร์คอมพิวเตอร์เพื่อให้ได้ผลลัพธ์ทันตามกำหนดเวลาที่ต้องการ งานที่ต้องใช้การประมวลผลปริมาณมากๆยังไม่ได้จำกัดอยู่แต่ในวงวิทยาศาสตร์และเทคโนโลยีเท่านั้น ในปัจจุบันได้มีการนำเอาซุปเปอร์คอมพิวเตอร์ไปใช้ในการประมวลผลทางธุรกิจด้วย แต่กระนั้นก็ตามข้อจำกัดของซุปเปอร์คอมพิวเตอร์ยังมีอยู่คือ ยังมีการใช้งานอยู่แต่ในวงแคบๆ เนื่องจากซุปเปอร์คอมพิวเตอร์มีราคาแพง (หลายร้อยล้านบาท) จึงมีเพียงบางองค์กรเท่านั้นที่จะมีเครื่องระดับซุปเปอร์คอมพิวเตอร์ใช้งาน

เมื่อมองถึงแนวโน้มของการใช้เครื่องคอมพิวเตอร์จากอดีตจนถึงปัจจุบัน แต่ก่อนการใช้งานคอมพิวเตอร์ส่วนใหญ่จะอยู่แต่ในองค์กรธุรกิจที่มีเงินทุนมากพอ ซึ่งแต่ก่อนนั้นจะมีแต่เครื่องคอมพิวเตอร์แบบเมนเฟรมอย่างเดียวเท่านั้น ต่อมาได้มีการใช้งานเครื่องมินิคอมพิวเตอร์ในหลายๆองค์กรมากขึ้น ปัจจุบันเครื่องคอมพิวเตอร์ส่วนบุคคลหรือไมโครคอมพิวเตอร์ได้มีบทบาทอย่างมาก ทั้งในด้านธุรกิจ วิทยาการและความบันเทิง องค์กรส่วนมากได้มีการนำเอาเครื่องระดับไมโครคอมพิวเตอร์มาช่วยในการใช้งาน เนื่องจากความสามารถของเครื่องไมโครคอมพิวเตอร์ได้ทวีขึ้นไปอย่างมาก ซึ่งจะสวนทางกับราคาของมันที่นับวันจะถูกลงเรื่อยๆ หรือพูดได้ว่าประสิทธิภาพต่อราคาของเครื่องไมโครคอมพิวเตอร์ดีขึ้นและดีกว่าเครื่องซุปเปอร์ด้วยซ้ำ เพราะตลาดในปัจจุบันเน้นที่จะพัฒนาไมโครคอมพิวเตอร์ซึ่งเป็นเครื่องระดับบุคคลซึ่งเป็นตลาดใหญ่ที่สุดของคอมพิวเตอร์ การผลิตปริมาณมากๆย่อมทำให้ราคาถูกลงตามไป

ด้วย ดังนั้นจึงมีแนวความคิดที่จะนำเอาไมโครคอมพิวเตอร์เหล่านี้ซึ่งมีประสิทธิภาพต่อราคาสูง มาต่อเข้าด้วยกันผ่านระบบเครือข่ายและมองระบบโดยรวมเป็นเสมือนก้อนๆหนึ่ง ซึ่งการรวมเครื่องไมโครคอมพิวเตอร์หลายๆเครื่องเข้าเป็นระบบเดียวนั้น ทำให้มีความสามารถเทียบเท่ากับเครื่องซุปเปอร์คอมพิวเตอร์เลยทีเดียว

ในปัจจุบันมีระบบคลัสเตอร์อยู่มากมายหลายแบบ แต่ที่นิยมและแพร่หลายมากที่สุดคือคลัสเตอร์แบบเบียวูล์ฟ (Beowulf Class) ซึ่งทำงานบนแพลตฟอร์มลินุกซ์ และใช้ระบบไฟล์ข้อมูลแบบเอ็นเอฟเอส (NFS ,Network File System) คือเครื่องลูกข่าย (node) ในระบบ ไม่จำเป็นต้องมีหน่วยความจำสำรอง (secondary storage device) ระบบไฟล์ทั้งหมดจะถูกเก็บอยู่ที่เซิฟเวอร์ เมื่อระบบไฟล์ถูกเก็บอยู่รวมกันที่เดียวทำให้ง่ายต่อการจัดการ และเครื่องลูกข่ายก็ไม่จำเป็นต้องใช้หน่วยความจำสำรอง ทำให้ลดค่าใช้จ่ายลงไปได้เยอะ ข้อดีอีกอย่างหนึ่งของการใช้แพลตฟอร์มลินุกซ์คือ ระบบใช้ทรัพยากรของระบบน้อย เครื่องระดับ i386 หน่วยความจำ (RAM) 8 Mb ก็เพียงพอที่จะรันลินุกซ์ได้แล้ว ทำให้การทำงานรวดเร็วกว่าแพลตฟอร์มอื่นๆ ข้อดีอีกประการหนึ่งคือ สามารถหาระบบปฏิบัติการมาใช้ได้ฟรี และในวิทยานิพนธ์ฉบับนี้ ก็จะยึดหลักการจัดการแบบเบียร์วูล์ฟเป็นสำคัญ

1.2 วัตถุประสงค์ของงานวิจัย

1.2..1 ศึกษาการทำงานของระบบประมวลผลแบบกลุ่ม (cluster computing) ทั้งแบบโดยรวมคือส่วนที่เป็นความคิดหลักๆของคลัสเตอร์ทั่วๆไป และศึกษาเจาะรายละเอียดของคลัสเตอร์แบบเบียร์วูล์ฟ

1.2.2 ศึกษาความเป็นไปได้ในการสร้างระบบขึ้นมาใช้งานจริง โดยดูจากอุปกรณ์ที่มีอยู่ และสร้างระบบตัวอย่างขึ้นมาให้สามารถใช้งานได้จริง

1.2.3 ศึกษาการเขียนโปรแกรมเชิงขนาน (parrallel programming) เพื่อทดสอบการทำงานบนระบบคลัสเตอร์ที่เราสร้างขึ้นมา

1.2.4 สร้างเครื่องมือเฝ้าติดตามการทำงานของระบบ (monitoring tool) เพื่อช่วยให้การจัดการระบบ รวมถึงการทดสอบและรันโปรแกรมเชิงขนาน เป็นไปได้โดยง่ายดาย โดยเน้นให้ผู้ใช้งานไม่จำเป็นมีความรู้ระบบมากนัก โดยเครื่องมือที่เราสร้างขึ้นจะช่วยอำนวยความสดวกให้

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้จะเป็นการสร้างระบบตัวอย่างขึ้นมาใช้งานโดยเราจะไม่เน้นประสิทธิภาพ (performance) มากนัก เนื่องจากข้อจำกัดทางด้านฮาร์ดแวร์และงบประมาณ แต่เราจะสร้างระบบที่สามารถใช้งานได้จริง และสามารถขยายระบบเพิ่มเติมได้ในอนาคตถ้ามีงบประมาณเพียงพอ งานวิจัยนี้นอกจากจะศึกษาการทำงานและสร้างคลัสเตอร์ขึ้นมาใช้จริงแล้ว เรายังได้ทดสอบระบบโดยการนำโปรแกรมเชิงขนานมารันบนระบบของเราด้วย และส่วนสุดท้ายที่สำคัญคือการสร้างเครื่องมือที่ช่วยอำนวยความสดวกในการจัดการคลัสเตอร์สำหรับผู้ใช้งานทั่วไป (user) และผู้จัดการระบบ (administrator) ในงานวิจัยนี้เราได้ประสบปัญหาโดยตัวเองจากการทำงานจริงบนระบบของเรา จึงได้มีจุดประสงค์เพื่อที่จะแก้ไขปัญหานั้นๆที่เราได้พบมา และปัญหาที่คาดว่าน่าจะเกิดกับผู้อื่นด้วย ซึ่งก็อาจไม่สามารถอำนวยความสดวก

สำหรับคนทุกคนได้ เป้าหมายหลักของเครื่องมือของเราคือ ช่วยลดความยุ่งยากซับซ้อนของการจัดการระบบคลัสเตอร์รวมถึงการเฝ้าติดตามการทำงานของระบบด้วย โดยผู้ใช้งานไม่จำเป็นต้องมีความรู้คำสั่งระดับล่างๆมากนัก โดยผู้ใช้งานจะเน้นการเขียนโปรแกรมเชิงขนานแล้วนำโปรแกรมที่ได้มาทดสอบกับระบบคลัสเตอร์ของเราโดยมีเครื่องมือที่เราสร้างขึ้นคอยจัดการคำสั่งระดับล่างๆแทนผู้ใช้งาน

1.4 วิธีการดำเนินงาน

งานวิจัยนี้จะเริ่มต้นด้วยการศึกษาทฤษฎีและการทำงานของระบบคลัสเตอร์ ,ศึกษาข้อดีข้อเสียของคลัสเตอร์แบบต่างๆ ศึกษาการเขียนโปรแกรมเชิงขนาน แล้วเข้าสู่กระบวนการของการสร้างระบบขึ้นมาใช้งานจริง จากนั้นจึงนำโปรแกรมเชิงขนานมาทดลองรันบนระบบของเราเพื่อดูผลลัพธ์ที่ได้

ในส่วนของเครื่องมือเฝ้าติดตามการทำงานของระบบ เราจะเริ่มด้วยการศึกษาการเขียนโปรแกรมบนเอ๊กซ์วินโดว (X-Window) ศึกษาภาษา ทิกเคิล (TCL) และทีเค (TK) ซึ่งเป็นภาษาหนึ่งที่เหมาะกับการทำส่วนติดต่อกับผู้ใช้งาน (user interface) บนเอ๊กซ์วินโดว จากนั้นจึงวิเคราะห์ปัญหาที่เกิดขึ้นจากการใช้งานจริง รวมถึงหาวิธีที่จะทำให้การทำงานบนระบบคลัสเตอร์ของเราง่ายขึ้น สดวกขึ้น จากนั้นจึงเข้าสู่ขั้นตอนของการออกแบบโปรแกรมเฝ้าติดตามการทำงานระบบ ตามด้วยการสร้างโปรแกรมขึ้นมา และทดสอบในสภาพใช้งานจริงด้วย

และสุดท้ายจะเป็นการสรุปการทำงาน และผลงานที่ได้จากการวิจัยนี้ และแนวทางในการประยุกต์ใช้งานวิจัยนี้ รวมถึงแนวทางในการพัฒนาต่อด้วย

บทที่ 2

ความสามารถในการขยายขนาดและคลัสเตอร์

(Scalability and Clustering)

2.1 พัฒนาการของสถาปัตยกรรมคอมพิวเตอร์

Generation Period	Technology and Architecture	Software and Operating System	Representative System
First 1946-1956	Vacuum tubes and relay memory, single-bit CPU with accumulator-based instruction set	Machine/assembly languages, programs without subroutines	ENIAC, IBM701, Princeton IAS
Second 1956-1967	Discrete transistors, core memory, floating-point accelerator, I/O channel	Algol and Fortran with compilers, batch processing OS	IBM 7030, CDC 1604, Univac LARC
Third 1967-1978	Integrated circuits, pipelined CPU, microprogrammed control unit	C language, multiprogramming, Timesharing OS	PDP-11 IBM 360/370 CDC 6600
Fourth 1978-1989	VLSI microprocessors, solid-state memory, multiprocessors, vector supercomputers	Symmetric multiprocessing, parallelizing compilers, message-passing libraries	IBM PC, VAX 9000, Cray X/MP
Fifth 1990-present	ULSI circuits, scalable parallel computers, workstation clusters, Intranet, Internet	Java, microkernels, Multithreading, distributed OS, World-Wide-Web	IBM SP2, SGI Origin 2000, Digital TruCluster

ตารางที่ 2-1 แสดงพัฒนาการของคอมพิวเตอร์ในยุคต่างๆ

จากประวัติศาสตร์คอมพิวเตอร์ได้เริ่มมีขึ้นมาประมาณ 50 ปีแล้ว และได้มีการพัฒนาต่อกันมาเรื่อยๆ จนถึงปัจจุบัน เมื่อเราย้อนกลับไปดูคอมพิวเตอร์ในยุคแรกๆสมัยนั้นยังใช้หลอดสูญญากาศ และหน่วยความ

จำแบบรีเลย์ (relay memory) หน่วยประมวลผลสมัยนั้นเป็นแบบ 1 บิทเท่านั้น ต่อมาได้มีการพัฒนาอุปกรณ์สารกึ่งตัวนำขึ้น ทำให้เกิดการนำทรานซิสเตอร์ขึ้นมาใช้งานแทนหลอดสูญญากาศ มีการเพิ่มการประมวลผลทศนิยม และได้มีการสร้างส่วนติดต่อกับอุปกรณ์ภายนอกเพิ่มขึ้น ในยุคที่สามได้มีการรวมเอาวงจรขนาดใหญ่ๆบรรจุเข้าไว้ในตัวถังเดียวกัน เรียกว่าไอซี (IC) ซึ่งทำให้วงจรมีขนาดเล็กลงมาก หน่วยประมวลผล เริ่มจะมีการใช้ไปป์ไลน์ ในยุคที่สี่มีการใช้เทคโนโลยี VLSI ซึ่งภายในจะบรรจุทรานซิสเตอร์ไว้จำนวนมากมาย ทำให้ซีพียูมีวงจรต่างๆภายในเพิ่มขึ้น ความสามรถเพิ่มขึ้นในขณะที่ขนาดเล็กลง และในยุคนี้ระบบหลายหน่วยประมวลผล (multiprocessors) และซุปเปอร์คอมพิวเตอร์ได้ถือกำเนิดขึ้นมา ในยุคที่ห้าได้มีการพัฒนาไอซีต่อไปอีกเป็น ULSI (Ultra-Large-Scale-Integrate-Circuit) ซึ่งภายในจะบรรจุทรานซิสเตอร์ไว้จำนวนมหาศาล ยกตัวอย่างเช่น ในปี ค.ศ. 1997 ได้มีการสร้างโปรเซสเซอร์ที่บรรจุทรานซิสเตอร์ไว้ถึง 10 ล้านตัว และมี DRAM ขนาด 256 MB ขึ้นเป็นต้น

เมื่อมองดูในด้านซอฟต์แวร์ในยุคแรกๆจะเป็นการใช้ภาษาเครื่อง (machine code) และภาษาแอสเซ็มบลี (assembly) เป็นส่วนใหญ่ ในยุคต่อมาเริ่มมีภาษาระดับสูงเพิ่มขึ้นเช่นภาษาฟอร์แทรน (Fortran) และ compiler ต่างๆขึ้น ในยุคที่สามภาษาชั้นสูงต่างๆเกิดขึ้นมากมายเช่นภาษาซี (C) ในยุคนี้เริ่มมีระบบปฏิบัติการที่สามารถรันงานได้พร้อมกันหลายๆงาน และเกิดระบบที่เรียกว่าการแบ่งเวลา (time sharing) ของระบบปฏิบัติการเกิดขึ้น ในยุคที่สี่มีการพัฒนาฮาร์ดแวร์ที่ใช้หลายโปรเซสเซอร์ รวมถึงได้มีการพัฒนาการโปรแกรมเชิงขนานเกิดขึ้น และในยุคที่ห้าได้เกิดการพัฒนาในการประมวลผลไปอย่างมากจนเกิดการประมวลผลแบบกลุ่มหรือคลัสเตอร์ซึ่งเป็นส่วนของวิทยานิพนธ์เล่มนี้ครอบคลุมอยู่ด้วย

2.2 สถาปัตยกรรมคอมพิวเตอร์ที่สามารถขยายขนาดได้ (Scalable Computer Architectures)

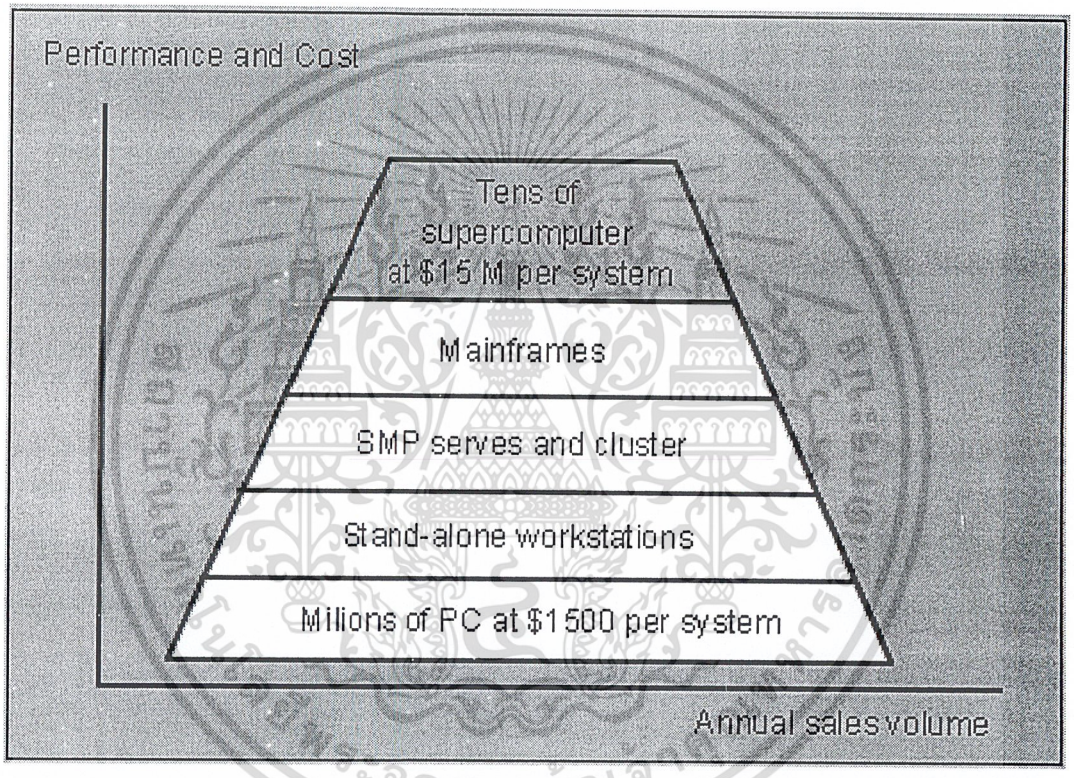
2.2.1 คำนิยาม

สถาปัตยกรรมคอมพิวเตอร์ที่สามารถขยายขนาดได้ (scalable Computer Architectures) คือระบบคอมพิวเตอร์ที่ประกอบด้วยฮาร์ดแวร์และซอฟต์แวร์ ที่สามารถเพิ่มขนาดให้ใหญ่ขึ้นเพื่อเพิ่มประสิทธิภาพโดยรวม และยังสามารถลดขนาดระบบลงได้เพื่อลดต้นทุนทางด้านราคา โดยการเพิ่มขนาดและลดขนาดนั้นจะเกิดผลกระทบโดยรวมกับระบบในลักษณะต่างๆดังต่อไปนี้

- ประสิทธิภาพ โดยทั่วไปแล้วผู้ใช้งานคาดหวังว่าเมื่อเพิ่มทรัพยากรของระบบขึ้นเป็นจำนวน n เท่า แล้วระบบควรจะมีความสามารถเพิ่มขึ้น n เท่าด้วย
- ราคา ระบบที่ดีนั้น การเพิ่มขึ้นของระบบเป็นจำนวน n เท่า ไม่ควรจะมีราคาเพิ่มขึ้นมากกว่า n เท่า
- ความเข้ากันได้กับระบบเดิม เมื่อเราเพิ่มหรือลดขนาดของระบบแล้ว ด้วยเหตุผลใดๆก็ตาม ผู้ใช้งานต้องไม่ได้รับผลกระทบมากนัก หรือทางที่ดีผู้ใช้งานไม่ควรจะต้องเปลี่ยนวิธีการทำงานไปเมื่อระบบเปลี่ยนไป เช่นผู้ใช้งานอาจจะใช้โคด (code) ตัวเดิม , ระบบปฏิบัติการตัวเดิม กับระบบใหม่ได้แต่สิ่งที่ผู้ใช้งานจะได้เพิ่มขึ้นก็คือความเร็ว เป็นต้น

2.2.2 ปิรามิดคอมพิวเตอร์

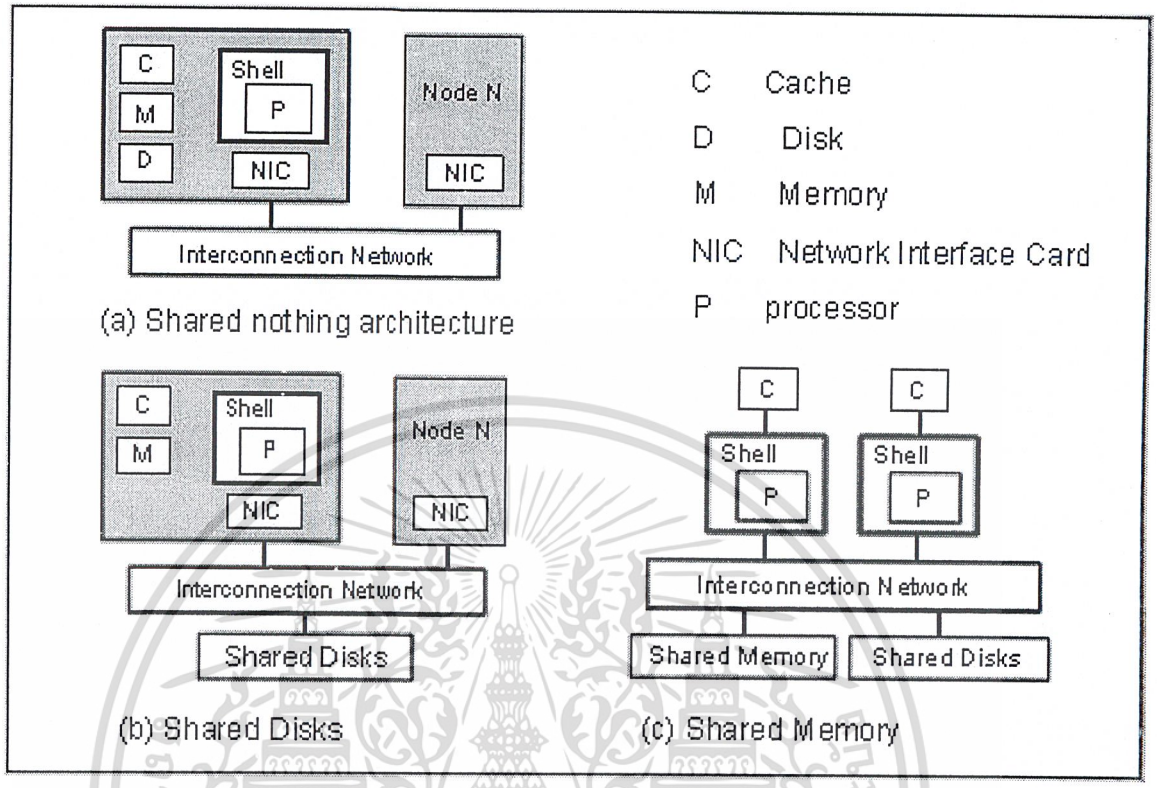
ในตลาดคอมพิวเตอร์ในปัจจุบัน ได้มีการแบ่งประเภทของคอมพิวเตอร์ออกเป็นกลุ่มใหญ่ๆได้ดังรูปที่ 2.1 ซึ่งจะอธิบายรายละเอียดของประเภทของคอมพิวเตอร์สัมพันธ์กับจำนวนการใช้งานที่มีอยู่จริง ในชั้นบนสุดจะเป็นซุปเปอร์คอมพิวเตอร์ที่มีราคาแพง (หลายร้อยล้านบาท) ประสิทธิภาพสูงสุด แต่มีจำนวนการใช้งานเพียงไม่กี่สิบเครื่องเท่านั้น ถัดลงมาจะเป็นเครื่องเมนเฟรม (mainframe), SMP ,workstation ซึ่งมีประสิทธิภาพรองๆลงมาในขณะที่ราคาถูกลงเรื่อยๆ และมีจำนวนการใช้งานมากขึ้น ในชั้นล่างสุดจะเป็นพีซี ซึ่งมีจำนวนผู้ใช้งานสูงสุด ประสิทธิภาพต่ำกว่าแบบอื่น แต่ราคาก็ถูกด้วย



รูปที่ 2-1 ปิรามิดของประเภทคอมพิวเตอร์

ในการสร้างระบบคอมพิวเตอร์แบบขยายขนาดได้ เราจะพิจารณาใช้คอมพิวเตอร์ประเภทต่างๆ ในปิรามิด ขึ้นอยู่กับปัจจัยหลายๆประการเช่น งบประมาณมีมากน้อยแค่ไหน ต้องการประสิทธิภาพโดยรวมเป็นอย่างไร ความเข้ากันได้กับระบบมีมากน้อยแค่ไหน ซึ่งในวิทยานิพนธ์ฉบับนี้จะเน้นที่การใช้เครื่องคอมพิวเตอร์พีซี ซึ่งมีราคาต่อหน่วยต่ำ และมีปริมาณการใช้มากสุด ในขณะที่ประสิทธิภาพเทียบกับราคาอยู่ในขั้นสูง มาประกอบกันเป็นระบบคลัสเตอร์ ซึ่งก็เป็นสถาปัตยกรรมคอมพิวเตอร์ขยายขนาดได้รูปแบบหนึ่ง ดังจะได้อธิบายในส่วนต่อไป

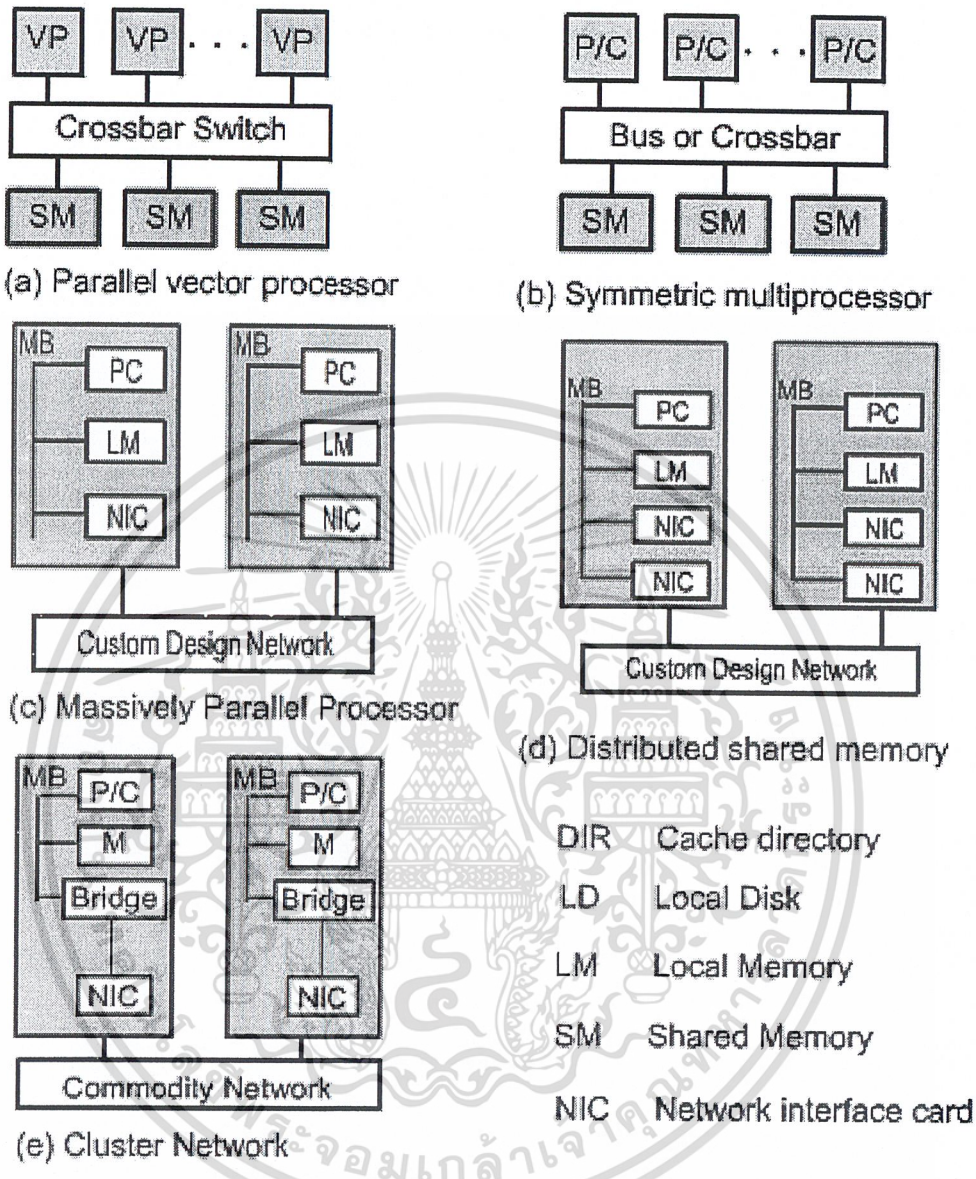
2.2.3 สถาปัตยกรรมโดยทั่วไปของระบบคอมพิวเตอร์ที่สามารถขยายขนาดได้



รูปที่ 2-2 รูปการจัดสถาปัตยกรรมโดยทั่วไปของระบบคอมพิวเตอร์ที่สามารถขยายขนาดได้

จากรูปที่ 2.2จะเป็นแนวความคิดหลักๆของการจัดโครงสร้างสถาปัตยกรรมแบบสามารถขยายขนาดได้ซึ่งเป็นการเชื่อมต่อของคอมพิวเตอร์หลายๆเครื่องในแบบขนานผ่านเครือข่าย โดยเครื่องคอมพิวเตอร์แต่ละเครื่องจะถูกเรียกว่าโหนด (node) ในแต่ละโหนดจะประกอบไปด้วย แคช (C),ดิสค์ (D), หน่วยความจำ (M), เครือข่าย (NIC), และสุดท้าย โปรเซสเซอร์ (P) เมื่อระบบถูกรวมเข้าด้วยกันจากหลายๆโหนด ดังนั้นเพื่อเพิ่มความคล่องตัวและความสามารถของระบบ จึงได้มีการใช้ทรัพยากรของระบบบางส่วนร่วมกัน เช่นดิสค์หรือหน่วยความจำเป็นต้น

2.2.4 แบบจำลองทางกายภาพของระบบคอมพิวเตอร์ขนานแบบต่างๆ



รูปที่ 2-3 แบบจำลองทางกายภาพของระบบคอมพิวเตอร์แบบขนาน ซึ่งประกอบด้วย PVP, SMP, MPP, DSM และ COW (cluster)

- **Parallel Vector Processors** เป็นระบบของเครื่องคอมพิวเตอร์ที่มีประสิทธิภาพสูงมากๆเช่น เครื่อง Cray C-90, Cray T-90 และ NEC SX-4 เป็นต้น ภายในจะประกอบไปด้วยเวคเตอร์โปรเซสเซอร์ (vector processor) ที่มีประสิทธิภาพสูงมาก (มากกว่า 1Gflops/s) จำนวนหลายๆตัวต่อเข้าด้วยกันผ่านระบบเครือข่ายที่สร้างขึ้นมาโดยเฉพาะ ซึ่งเป็นเครือข่ายความเร็วสูงมาก และมี

การใช้หน่วยความจำร่วม (share memory) ซึ่งมีอัตราการส่งข้อมูลที่สูงมาก ยกตัวอย่างเช่นเครื่องคอมพิวเตอร์ T-90 สามารถส่งผ่านข้อมูลจากหน่วยความจำได้ถึง 14 GB/s

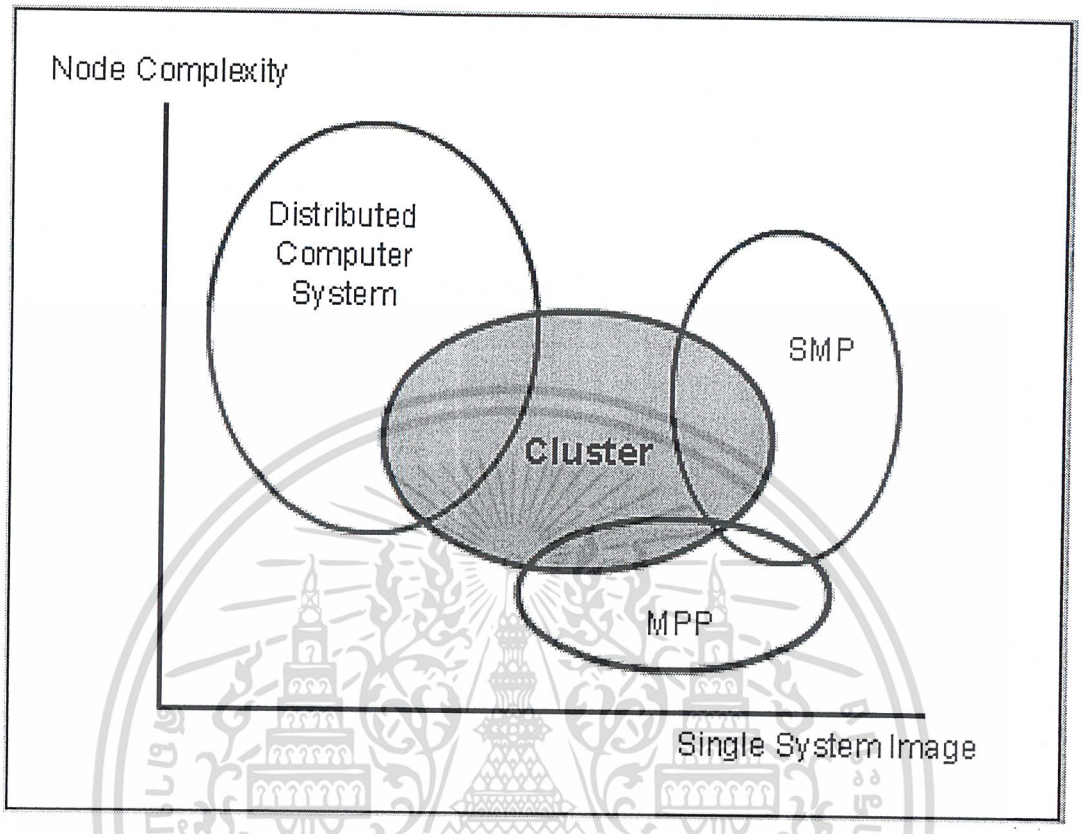
- **Symmetric Multiprocessors** ยกตัวอย่างเช่นเครื่อง IBM R50 , DEC Alpha server 8400 เป็นต้น เครื่องคอมพิวเตอร์แบบเอสเอ็มพีนี้จะประกอบไปด้วยโปรเซสเซอร์จำนวนหลายๆตัว ต่อเข้าด้วยกันผ่านเครือข่ายที่ออกแบบขึ้นโดยเฉพาะ และมีหน่วยความจำร่วม ระบบแบบเอสเอ็มพีนี้มีความแตกต่างจาก พีวีพี (PVP) คือ เอสเอ็มพีจะใช้โปรเซสเซอร์ที่มีขายในตลาดทั่วไป ในขณะที่พีวีพีจะใช้เวคเตอร์โปรเซสเซอร์ ซึ่งออกแบบมาโดยเฉพาะ ในขณะที่เอสเอ็มพี จะมีความสมมาตรของโปรเซสเซอร์มากกว่า คือโปรเซสเซอร์ทุกตัวจะมีลำดับความสำคัญในการเข้าถึงหน่วยความจำเท่ากัน และด้วยความสมมาตรนี้เองทำให้ระบบสามารถเพิ่มโปรเซสเซอร์ขึ้นไปได้หลายสิบตัว ยกตัวอย่างเช่นเครื่อง Sun Ultra Enterprise 10000 มีจำนวนโปรเซสเซอร์ถึง 64 ตัว และระบบแบบเอสเอ็มพี ยังแพร่หลายในการใช้งานทางธุรกิจมากกว่าอีกด้วย

- **Massively Parallel Processor** ระบบแบบเอ็มพีพี (MPP Massively Parallel Processors) นี้มีความสามารถขยายขนาดสูงมาก ในบางระบบสามารถเพิ่มจำนวนโปรเซสเซอร์ได้หลายพันตัว โดยการต่อเชื่อมกันผ่านเครือข่ายที่ออกแบบขึ้นมาโดยเฉพาะและมีการใช้หน่วยความจำแบบกระจาย (distributed memory) ซึ่งเหมาะกับงานบางประเภทเช่น การคำนวณทางวิทยาศาสตร์, การจำลองทางวิศวกรรมศาสตร์ เป็นต้น ลักษณะเด่นๆของระบบแบบ เอ็มพีพีนี้ก็คือ มีการใช้โปรเซสเซอร์ที่มีขายในตลาด, ใช้หน่วยความจำแบบกระจาย, เชื่อมต่อกันผ่านเครือข่ายความเร็วสูง เป็นต้น

- **Distributed Shared-Memory** ยกตัวอย่างเช่นเครื่องคอมพิวเตอร์ Cray T3D ระบบโดยรวมจะคล้ายๆเอ็มพีพี แต่จะมีการใช้แคชไดเรกทอรี (catch directory) และจุดเด่นๆของระบบแบบนี้เทียบกับแบบเอสเอ็มพีคือ จะมีการกระจายหน่วยความจำออกไปยังโหนดต่างๆอย่างชัดเจน

- **Cluster** ประกอบไปด้วยโหนดที่สามารถคำนวณได้ (computation node) อาจจะเป็นพีซีหรือเวอร์กสเตชันจำนวนหลายๆเครื่องต่อเข้าด้วยกันผ่านเครือข่ายที่เป็นมาตรฐานอยู่แล้วเช่น Ethernet FDDI ATM เป็นต้น โดยแต่ละโหนดจะมีหน่วยความจำสำรองของตัวเอง และมีระบบปฏิบัติการที่สมบูรณ์ในแต่ละโหนดอยู่แล้ว ซึ่งก็หมายความว่าแต่ละโหนดสามารถทำงานได้เป็นเอกเทศ แต่เราจะต้องสร้างซอฟต์แวร์ขึ้นมาตรงกลางระหว่างซอฟต์แวร์กับระบบปฏิบัติการเพื่อให้ซอฟต์แวร์มองทั้งระบบเสมือนเป็นหนึ่งเดียวกัน ซึ่งวิทยานิพนธ์เล่มนี้ก็จะคลอบคลุมเนื้อหาตรงส่วนของระบบแบบคลัสเตอร์นี้เป็นสำคัญ

2.2.5 เปรียบเทียบข้อดีข้อเสียของสถาปัตยกรรมแบบต่างๆ



รูปที่ 2-4 เปรียบเทียบความซับซ้อนและความเป็นหนึ่งเดียวของระบบของสถาปัตยกรรมแบบต่างๆ

สถาปัตยกรรมแบบคลัสเตอร์, เอสเอ็มพี, เอ็มพีพี และ ระบบกระจาย (Distributed system) มีความแตกต่างกันในองประกอบของความซับซ้อนของระบบ และความเป็นหนึ่งเดียวของระบบ แต่ก็มีบางส่วนที่ซ้อนทับกัน หรือมีส่วนที่คล้ายกันอยู่บ้าง ระบบกระจายจะเป็นรูปแบบหนึ่งของสถาปัตยกรรมที่สามารถขยายขนาดได้ที่มีความซับซ้อนของระบบสูงกว่าแบบเอ็มพีพี เพราะมีการกระจายหน่วยความจำออกไปตามโหนดและยังมีพื้นที่สำหรับเก็บข้อมูลแยกกันด้วย ในขณะที่เอสเอ็มพีจะมีความเป็นหนึ่งเดียวของระบบมากกว่าโดยโปรเซสเซอร์ทั้งหมดจะใช้หน่วยความจำพื้นที่เดียวกันและมีการใช้บัสร่วมกัน ในขณะที่คลัสเตอร์จะอยู่ตรงกลางๆ คือเป็นระบบที่มีความซับซ้อนพอสมควรเนื่องจากแต่ละโหนดจะมีหน่วยความจำและพื้นที่เก็บข้อมูลของตัวเอง ซ้ำยังติดต่อกันผ่านเครือข่าย ทำให้ความเป็นหนึ่งเดียวของระบบลดลงไป

System Characteristic	MPP	SMP	Cluster	Distributed System
Number of nodes (*N*)	100-1000	10	100	10-1000
Node Complexity	Fine or medium grain	Medium or coarse grain	Medium grain	Wide range
Internode communication	Message passing or shared variables	Shared memory	Message passing	Shared files, RPC, message passing
Job scheduling	Single run queue	Single run queue	Multiple queues	Independent multiple queues
SSI support	Partially	Always	Desired	No
Node OS copies and Type	*N* microkernel	One	*N* (homogeneous)	*N* (heterogeneous)
Address space	Multiple	Single	Multiple	Multiple
Internode security	Unnecessary	Unnecessary	Required	Required
Ownership	One organization	One organization	One or more organization	Many Organization
Network protocol	Nonstandard	Nonstandard	Standard	Standard
System availability	Low	Low	Highly	Medium
Performance metric	Throughput and turnaround time	Turnaround time	Throughput and turnaround time	Response time

ตารางที่ 2-2 เปรียบเทียบข้อดีข้อเสียของสถาปัตยกรรมแบบ Cluster MPP SMP และ Distributed System (SSI: Single system image, RPC Remote procedure call)

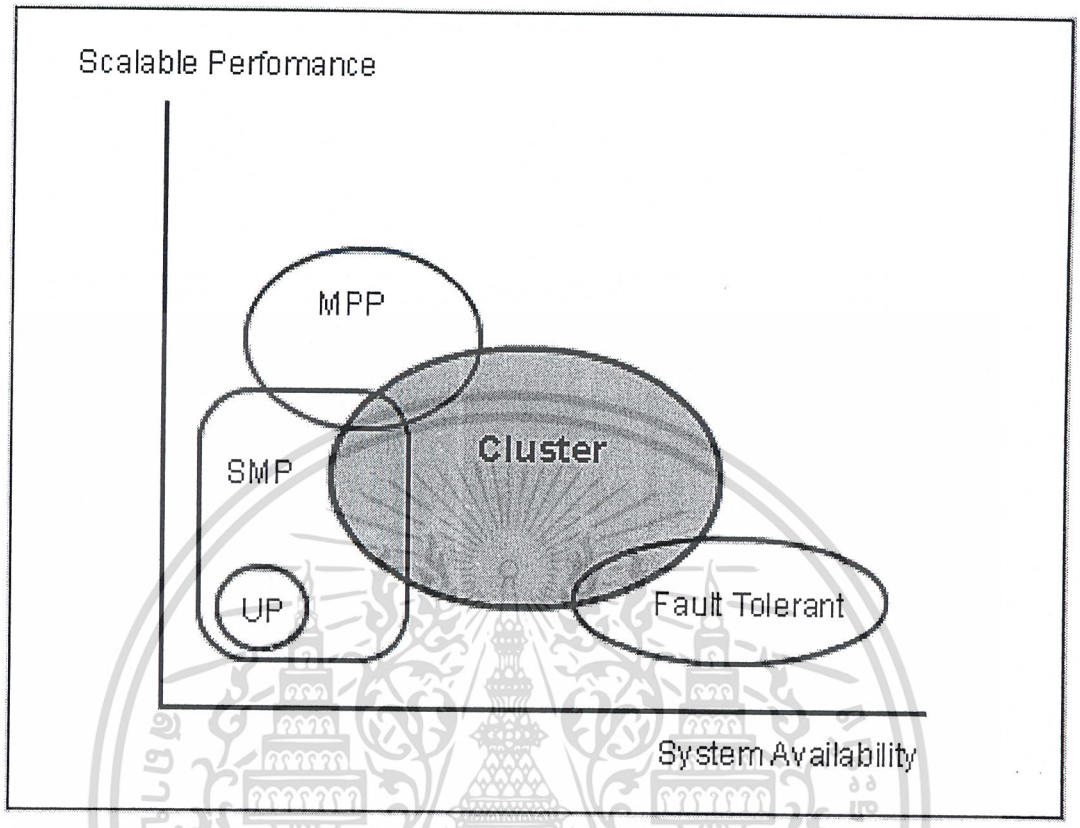
จากตารางที่ 2.2 เป็นการเปรียบเทียบกันระหว่างสถาปัตยกรรมที่นิยมใช้กันแพร่หลายได้แก่ เอ็มพีพี, เอสเอ็มพี, คลัสเตอร์ และระบบกระจาย ซึ่งมีรายละเอียดดังนี้

- **Number of nodes (*N*)** คือจำนวนของโหนดในระบบ ซึ่งจะเห็นว่าแบบเอ็มพีพีมีมากที่สุด (อาจถึง 1000 โหนด) ในขณะที่คลัสเตอร์จะมีไม่เกิน 100 โหนดเท่านั้น
- **Node complexity** คือความซับซ้อนของโหนด

- **Internode communication** คือการติดต่อสื่อสารระหว่างโหนดภายใน โดยหลักๆจะใช้ message passing แต่ที่จะมีแปลกอยู่บ้างคือเอสเอ็มพี ซึ่งจะใช้หน่วยความจำร่วม ซึ่งติดต่อสื่อการผ่านบัสภายในอยู่แล้ว
- **Job scheduling** คือตารางเวลางาน ซึ่งจะกำหนดว่างานไหนทำก่อนหลัง ซึ่งเอ็มพีพีและเอสเอ็มพี จะเป็นแบบเดี่ยว ส่วนคลัสเตอร์และระบบกระจาย จะสามารถทำตารางงานได้หลายๆ อันพร้อมกัน
- **SSI support** คือการมองความเป็นหนึ่งเดียวของระบบ เอสเอ็มพี จะมีความเป็นหนึ่งเดียวของระบบอย่างชัดเจน ในขณะที่ระบบกระจายจะมีความเป็นอิสระของโหนด ทำให้ไม่มีความเป็นหนึ่งเดียวกัน เมื่อมองภาพโดยรวมของระบบ
- **Node OS copies and Type** คือระบบปฏิบัติการของระบบ จำนวนของระบบปฏิบัติการ และชนิด ยกตัวอย่างเช่น คลัสเตอร์จะมีจำนวน N โหนดที่มีระบบปฏิบัติการและทุกโหนดมีระบบปฏิบัติการชนิดเดียวกันหมด (homogeneous) ส่วนระบบกระจาย จะมีจำนวน N โหนดที่มีระบบปฏิบัติการเพียงแต่ว่า แต่ละโหนดนั้นอาจเป็นระบบปฏิบัติการคนละชนิดก็ได้ (heterogeneous) ส่วนเอสเอ็มพีนั้นมีการรันระบบปฏิบัติการเพียงหนึ่งเดียวเท่านั้น
- **Address space** คือการอ้างถึงพื้นที่ใช้งาน single คือทุกๆโหนดจะมีการอ้างพื้นที่ใช้งานที่เป็นหนึ่งเดียวและพื้นที่ใช้งานนั้นทุกๆโหนดสามารถใช้งานร่วมกันได้เสมือนเป็นของตัวเอง ส่วน multiple หมายถึงแต่ละโหนดสามารถอ้างพื้นที่ใช้งานของตัวเองได้และเป็นอิสระจากโหนดอื่นๆด้วย
- **Internode security** คือระบบรักษาความปลอดภัย ในแต่ละโหนด
- **Ownership** คือเจ้าของระบบ ซึ่งระบบแบบกระจายนั้น อาจจะเป็นไปได้ว่ามีการเชื่อมโยงกันหลายองค์กร และมีการใช้ทรัพยากรร่วมกัน ในขณะที่ระบบแบบอื่นๆ จะมีสถานที่ตั้งอยู่ในพื้นที่เดียวกัน ซึ่งก็อาจจะเป็นขององค์กรใดเพียงองค์กรหนึ่งเท่านั้น
- **Network protocol** คือมาตรฐานในการใช้เครือข่ายร่วมกัน จะเห็นว่าระบบแบบคลัสเตอร์และระบบแบบกระจาย จะมีการใช้มาตรฐานเครือข่ายที่มีอยู่แล้วเช่น Ethernet, FDDI, ATM เพื่อความง่าย และประหยัด ในขณะที่ระบบแบบเอสเอ็มพีและเอ็มพีพี จะมีการออกแบบระบบเครือข่ายขึ้นมาเฉพาะ เพื่อเพิ่มประสิทธิภาพให้สูงที่สุด
- **System availability** คือความสามารถในการทำงานของระบบ
- **Performance metric** คือวิธีในการวัดประสิทธิภาพโดยรวมของระบบ

2.3 ประสิทธิภาพของการขยายระบบ

เมื่อเราขยายระบบของเราออกไป สิ่งหนึ่งที่เราต้องพิจารณาคือความสามารถที่เพิ่มขึ้นนั้นมากน้อยแค่ไหนเมื่อเทียบกับส่วนประกอบต่างๆที่เพิ่มเข้ามาไม่ว่าจะเป็นโปรเซสเซอร์, หน่วยความจำ, หน่วยความจำสำรอง หรือแม้กระทั่งอุปกรณ์อินพุตเอาท์พุต (I/O) แต่ละระบบจะมีความยากง่ายต่างกันในการขยายขนาดของระบบ และประสิทธิภาพโดยรวมของระบบก็จะเพิ่มขึ้นต่างกันด้วย ดังแสดงในรูปที่ 2.5

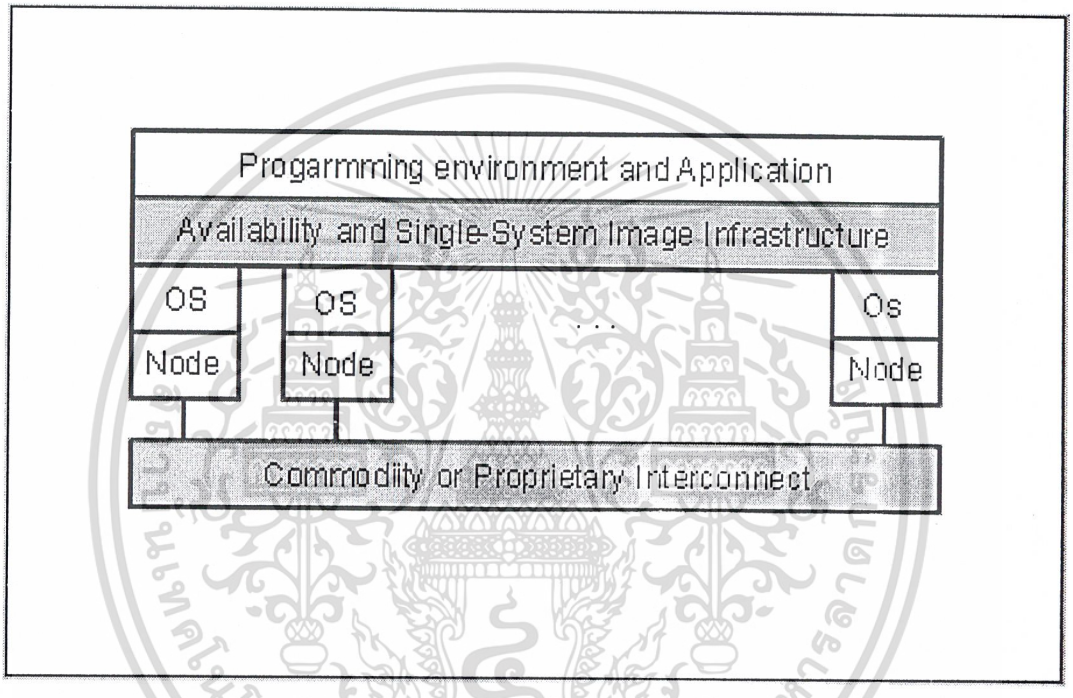


รูปที่ 2-5 เปรียบเทียบประสิทธิภาพในการขยายขนาดของระบบ

จากรูปที่ 2.5 จะแสดงรายละเอียดของระบบคอมพิวเตอร์แบบต่างๆเปรียบเทียบกันในด้านประสิทธิภาพในการขยายระบบและความสามารถของระบบ เมื่อดูที่ระบบแบบเอสเอ็มพีนั้นจะมีขีดจำกัดของการขยายระบบเนื่องจากการออกแบบระบบที่มีจำนวนโปรเซสเซอร์มากกว่า 4 ตัวขึ้นไปทำได้ยุ่งยาก เพราะต้องคำนึงถึงความสมมาตรของโปรเซสเซอร์ด้วย ในขณะที่ระบบคลัสเตอร์มีความสดวกกว่ามากในการเพิ่มโหนดเข้าไปในระบบ ส่วนระบบแบบ Fault Tolerant นั้น จะมีความเสถียรภาพกว่ามากเมื่อเทียบกับระบบแบบเอสเอ็มพี ซึ่งมีระบบปฏิบัติการเดียว และใช้หน่วยความจำร่วมกัน เมื่อเกิดข้อผิดพลาดขึ้นในระบบจะทำให้ทั้งระบบทำงานไม่ได้ ส่วนระบบเอ็มพีพีนั้น จะมองภาพรวมของทั้งระบบเป็นหนึ่งเดียว, การขยายขนาดนั้นทำได้อย่างดีเยี่ยมในขณะที่ ความสามารถของระบบจะเท่าๆกับแบบเอสเอ็มพี และขอดีอีกอย่างของคลัสเตอร์คือสามารถเพิ่มทั้งประสิทธิภาพโดยรวมและเสถียรภาพไปพร้อมๆกันได้ เมื่อมีการเพิ่มขนาดของระบบขึ้น

2.4 หลักการพื้นฐานของคลัสเตอร์

คลัสเตอร์คือการประกอบกันของโหนดที่สามารถคำนวณได้ (computation node) ซึ่งอาจจะเป็นคอมพิวเตอร์ส่วนบุคคล, เครื่องเวอร์กสเตชัน, หรือแม้กระทั่งเครื่อง เอสเอมพีเซิฟเวอร์ เข้าด้วยกันผ่านเครือข่ายความเร็วสูงเช่น LAN เพื่อเพิ่มประสิทธิภาพและความเสถียรภาพของระบบขึ้น ซึ่งโดยปกติแล้วแต่ละโหนดจะมีโปรเซสเซอร์,หน่วยความจำ,หน่วยความจำสำรอง และระบบปฏิบัติการซึ่งสามารถทำงานได้ด้วยตัวเองโดดๆอยู่แล้ว คลัสเตอร์จะเป็นการทำให้แต่ละโหนดเหล่านี้ ซึ่งมีระบบปฏิบัติการเป็นของตัวเอง, มีทรัพยากรของตัวเอง มารวมกันเข้าและมองภาพรวมของระบบให้เป็นเสมือนระบบเพียงหนึ่งเดียว เพื่อพร้อมที่จะรองรับสภาพการทำงานของโปรแกรมต่างๆดังได้แสดงในรูปที่ 2.6



รูปที่ 2-6 สถาปัตยกรรมของคลัสเตอร์

บทที่ 3
พื้นฐานการโปรแกรมเชิงขนาน
(Basic of Parallel Programming)

3.1 ภาพรวมของการโปรแกรมเชิงขนาน

ในปัจจุบันโปรแกรมเชิงขนานมีการพัฒนาไปน้อยมากเมื่อเทียบกับการพัฒนาทางด้านฮาร์ดแวร์ ซึ่งสนับสนุนการประมวลผลแบบขนาน อย่างไรก็ตามแนวโน้มของการพัฒนาโปรแกรมแบบขนานก็มีเพิ่มขึ้นเรื่อยๆเพราะการแยกงานออกเป็นส่วนย่อยๆมีข้อดีหลายอย่างดังจะได้กล่าวต่อไป

3.1.1 ข้อดีของโปรแกรมเชิงขนาน

- การแยกงานออกทำเป็นงานย่อยๆแบบขนาน ทำให้การประมวลผลเร็วขึ้น
- ในงานบางลักษณะ การโปรแกรมแบบขนานจะง่ายและเป็นธรรมชาติมากกว่า
- งานบางประเภท เมื่อทำเป็นโปรแกรมเชิงขนานแล้ว จะประมวลผลได้เร็วกว่าโปรแกรมเชิงลำดับในสภาพฮาร์ดแวร์ที่เท่ากัน
- มีฮาร์ดแวร์รองรับมากมาย

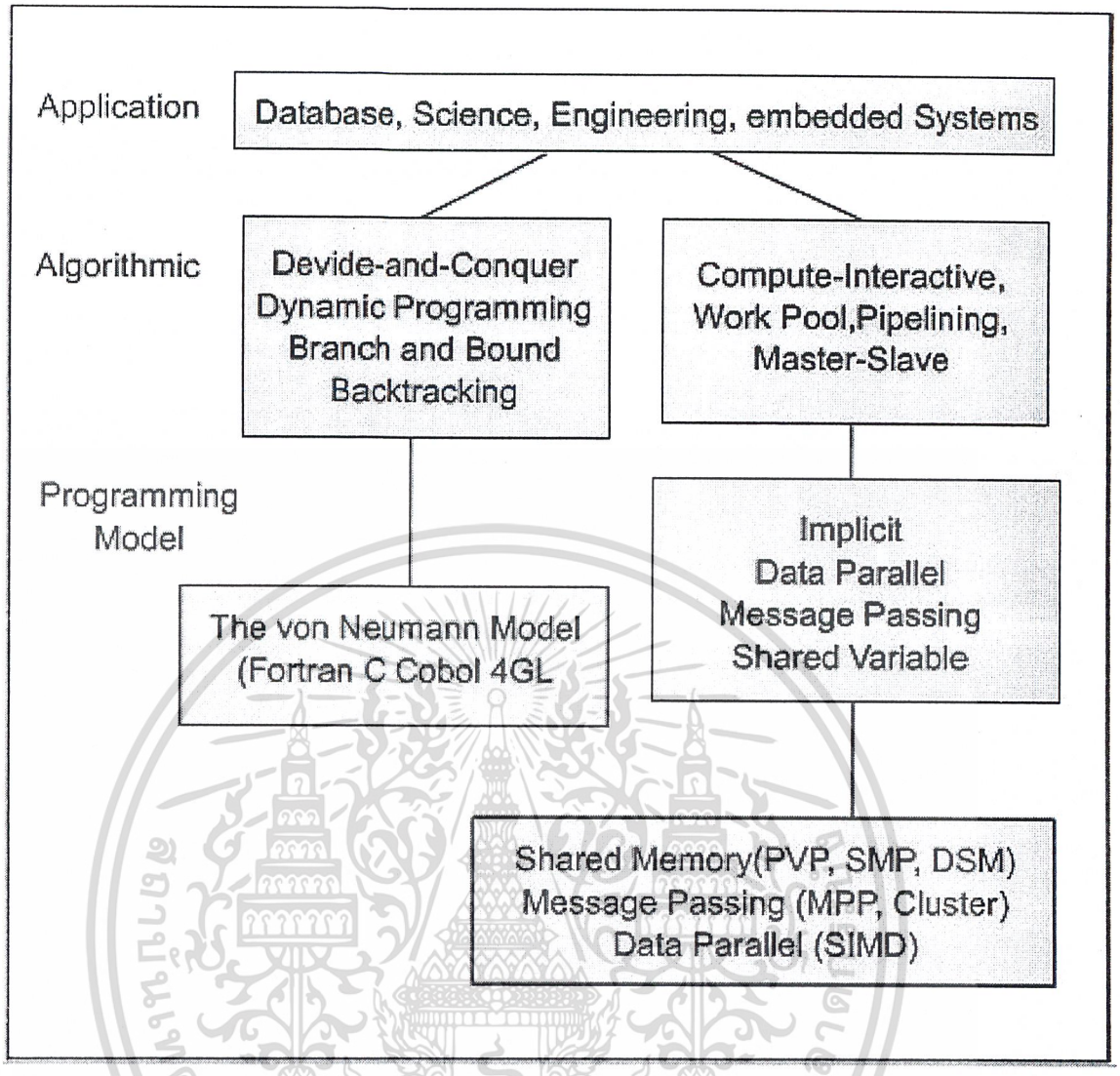
3.1.2 ความยุ่งยากของโปรแกรมเชิงขนาน

เนื่องจากการโปรแกรมเชิงขนานนั้นมีความยากลำบากกว่าการโปรแกรมเชิงลำดับ (sequential programming) มาก ซึ่งจะมองการโปรแกรมเชิงขนานก็คือการประกอบรวมกันของโปรแกรมเชิงลำดับขนาดย่อยๆนั้นเอง ซึ่งการที่จะโปรแกรมแบบขนานให้ดีได้นั้นจำเป็นต้องรู้หลักการการโปรแกรมเชิงลำดับอยู่ด้วย

อย่างที่สองคือ มีแบบจำลองการโปรแกรมเชิงขนานอยู่มากมายหลายแบบ ซึ่งแต่ละแบบนั้นก็จะแตกต่างกันไปในรายละเอียด ทำให้ไม่มีมาตรฐานในการใช้งานอย่างชัดเจน และการพัฒนาไม่เป็นไปในทิศทางเดียวกัน

อย่างที่สามคือ เครื่องมือตัวแปลภาษา (compiler) และโปรแกรมมสนับสนุนต่างๆของการโปรแกรมเชิงลำดับล้ำหน้ากว่าโปรแกรมสนับสนุนและตัวแปลภาษาของการโปรแกรมเชิงขนานมาก

เป็นเวลานานมาแล้วที่บุคคลทั่วไป มักจะฝึกเขียนโปรแกรมเชิงลำดับมากกว่าที่จะหัดเขียนโปรแกรมเชิงขนาน เพราะการเขียนโปรแกรมเชิงลำดับมีความเป็นธรรมชาติมากกว่า และมีการสะสมความรู้ของการโปรแกรมเชิงลำดับกันมาอย่างต่อเนื่อง ทำให้ค้นพบจุดบกพร่องต่างๆของการโปรแกรมเชิงลำดับและได้มีการปรับปรุงข้อบกพร่องนั้นกันมาอย่างต่อเนื่องด้วย ทำให้เราเปรียบเทียบการโปรแกรมแบบขนานและการโปรแกรมแบบลำดับได้ดังในรูปที่ 3.1



รูปที่ 3-1 เปรียบเทียบการโปรแกรมเชิงลำดับและการโปรแกรมเชิงขนาน

- การโปรแกรมเชิงลำดับ

เมื่อผู้ใช้งานต้องการโปรแกรมสำหรับใช้งานบนเครื่องคอมพิวเตอร์ทั่วๆไป จะพบว่ามีโปรแกรมให้เลือกใช้มากมาย โดยโปรแกรมเหล่านี้มีคนสร้างไว้ใช้งานอยู่แล้ว ซึ่งอาจจะมีโค้ดให้ใช้ด้วย ผู้ใช้เพียงแต่ทำการคอมไพล์ใหม่ให้เหมาะสมกับแพลตฟอร์มของเครื่องที่เราจะนำโปรแกรมไปใช้งานเท่านั้น หรือเมื่อผู้ใช้ต้องการพัฒนาโปรแกรมนั้นเพิ่มขึ้นไปอีกก็เพียงแต่หาอัลกอริทึมที่มีหลักการเข้ากับจุดประสงค์ของงานนั้นมาประยุกต์ใช้เข้ากับโคดที่มีอยู่แล้วนั้นได้เลย ในที่สุดแล้วก็ยังมีเครื่องมือสนับสนุนต่างๆให้เลือกใช้มากมาย

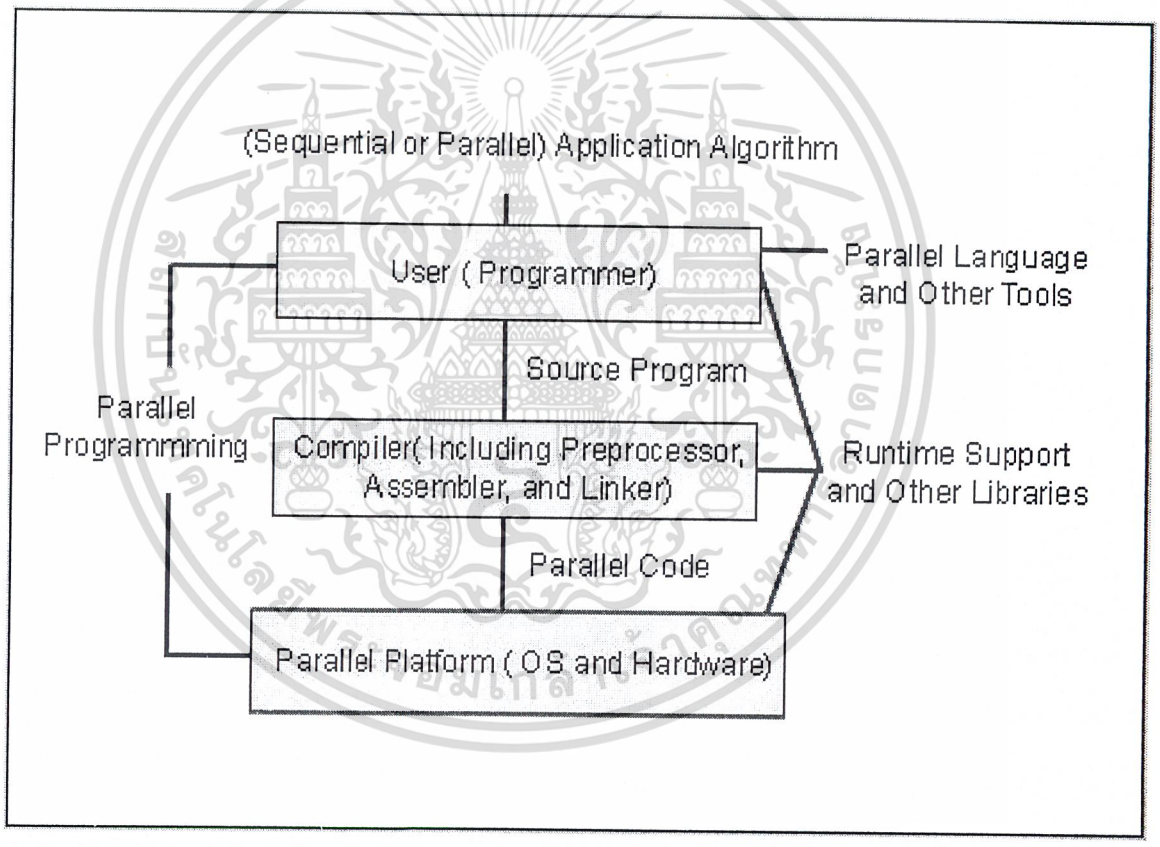
การโปรแกรมเชิงลำดับนั้นมีแบบจำลองเครื่องจักรคำนวณซึ่งออกแบบโดย วอน นิวแมน (Von Meumann) เป็นแบบจำลองซึ่งครอบคลุมอัลกอริทึมต่างๆของการโปรแกรมเชิงลำดับ และยังสามารถประยุกต์ใช้ได้กับทุกๆแพลตฟอร์มด้วย แบบจำลองของนิวแมนนั้นใช้กันมากว่า

40 ปีแล้ว และก็ยังใช้ได้ดีจนถึงปัจจุบัน มาตรฐานของภาษาต่างๆก็มีมากมายอาทิเช่น Fortran, Cobol, C เป็นต้น

- การโปรแกรมเชิงขนาน

การโปรแกรมเชิงขนานจะมีส่วนที่แตกต่างกับการโปรแกรมเชิงลำดับจุดหนึ่งคือ โคดของโปรแกรมที่ถูกสร้างขึ้นมาใช้งานกับสถาปัตยกรรมคอมพิวเตอร์แบบหนึ่ง จะไม่สามารถนำไปใช้กับสถาปัตยกรรมคอมพิวเตอร์แบบอื่นได้เลยยกตัวอย่างเช่น โปรแกรมที่รันบนเครื่องเอ็มพีพีไม่สามารถรันบนเครื่องแบบเอ็มพีพีได้ แบบจำลองการโปรแกรมเชิงขนานก็มีมากมายอาทิเช่น ดาต้าพาราเลล (Data Parallel), เมสเซจพาสซิ่ง (Message Passing), แชร์วาริเอเบิ้ล (Shared variable) ซึ่งในแต่ละแบบก็จะมีความแตกต่างกันดังจะได้อธิบายในส่วนต่อไป

3.1.3 องค์ประกอบของการคำนวณแบบขนาน



รูปที่ 3-2 องค์ประกอบของระบบประมวลผลแบบขนาน

ตามมุมมองของผู้ใช้งาน การประมวลผลแบบขนานนั้นมีขั้นตอนตามรูปที่ 3.2 ในขั้นตอนพัฒนาเมื่อเราเลือกที่จะใช้อัลกอริทึมใดๆในการแก้ปัญหาแล้ว ผู้ใช้งานต้องจะต้องแปลงอัลกอริทึมออกมาเป็นโปรแกรมโดยเขียนด้วยภาษาระดับสูงที่สนับสนุนการคำนวณแบบขนานด้วย

ขั้นต่อมาตัวแปลภาษาจะแปลงโคดที่เขียนขึ้นไปเป็นภาษาที่เครื่องคอมพิวเตอร์สามารถเข้าใจได้ทันที หรืออาจจะเรียกว่าภาษาเครื่องก็ได้ ซึ่งตัวแปลภาษานี้ต้องมีความสามารถในการแปลโคดต้นฉบับซึ่งเขียนออกมาเป็นแบบขนานให้สามารถรันบนแพลตฟอร์มแบบขนานได้อย่างมีประสิทธิภาพและส่วนประกอบอีกอย่างหนึ่งของการโปรแกรมแบบขนานคือ รันไทม์ซัพพอร์ต (Runtime Support) ซึ่งจะเป็นส่วนที่สนับสนุนการรันในเวลาใช้งานจริง ยกตัวอย่างรันไทม์ซัพพอร์ตในภาษา C เช่น “stdio” ซึ่งจะเป็นส่วนติดต่อกับการรับส่งขอมูลจากอุปกรณ์รอบข้าง เป็นต้น

3.1.4 หลักในการสร้างโปรแกรมเชิงขนาน

ในการที่จะได้มาซึ่งโปรแกรมเชิงขนานนั้น มีแบบจำลองหลักๆอยู่ 4 อย่างที่น่าสนใจคือ

- อิมพลิซิท (Implicit)
- ดาต้าพาราเลล (Data Parallel)
- เมสเซจพาสซิ่ง (Message Passing)
- แชร์วาริเอเบิล (Shared Variable)

ซึ่งจะบรรยายแบบละเอียดในส่วนต่อไป แต่ทั้งนี้การได้มาซึ่งโปรแกรมเชิงขนานโดยการใช้แบบจำลองเหล่านี้มีวิธีการหลักๆที่จะสร้างขึ้น 3 วิธีคือ

- **Library Subroutines** คือไลบรารี่ที่สนับสนุนการโปรแกรมเชิงขนานที่มีให้ใช้งานอยู่แล้ว เช่น MPI, PVM เป็นต้น
- **New Constructs** เป็นการสร้างจากตัวแปลภาษา โดยตัวแปลภาษานั้นจะออกแบบมาสำหรับการคอมไพล์โปรแกรมเชิงขนานโดยเฉพาะ เช่น “Fortran 90”
- **Compiler Directives** เป็นการจัดการของตัวคอมไพเลอร์โดยยังคงใช้ภาษาเดิมโค้ดตัวเดิม เพียงแต่ตัวแปลภาษาจะทำการคอมไพล์จากโปรแกรมเชิงลำดับไปเป็นโคดของโปรแกรมเชิงขนานให้อัตโนมัติ เช่น “pragmas”

Approach	Example	Advantage	Disadvantages
Library	MPI, PVM, Cray Craft	Easy to implement, Not need new compiler	No compiler check, analysis, optimization
New Constructs	Fortran 90, Cray Craft	Allow compiler check, Analysis, optimization	Difficult to implement, need a new compiler
Directives	HPF, Cray Craft	Between library and new constructs Can be ignored on a sequential platform	

ตาราง 3-1 เปรียบเทียบวิธีในการสร้างโปรแกรมเชิงขนาน

จากตารางที่ 3.1 จะเป็นการเปรียบเทียบข้อดีข้อเสียของวิธีการสร้างโปรแกรมเชิงขนานแบบต่างๆ จะเห็นว่าการใช้วิธีไลบรารีนั้นจะมีข้อดีคือ ง่ายในการนำไปใช้และไม่จำเป็นต้องสร้างคอมไพเลอร์ใหม่ ซึ่งวิทยานิพนธ์เล่มนี้เลือกที่จะใช้วิธีนี้ด้วย ส่วนรูปที่ 3.3 จะแสดงโค้ดที่ต่างกันของวิธีการทั้งสามแบบ

```
for ( i = 0 ; i < N ; i++) A[i] = b[i] * b[i+1];
for ( i = 0 ; i < N ; i++) c[i] = A[i] + A[i+1];
```

(a) A sequential code fragment

```
id = my_process_id ();
p = number_of_processor () ;
for ( i = id ; i < N ; i = i+p ) A[i] = b[i] * b[i+1] ;
barrier () ;
for ( i = id ; i < N ; i = i+p ) c[i] = A[i] + A[i+1] ;
```

(b) Equivalent parallel code using library routines

```
my_process_id(), number_of_process(), and barrier()
A(0:N-1) = b(0:N-1)*b(1:N)
c = A(0:N-1) + A(1:N)
```

(c) Equivalent code in Fortran 90 using array operations

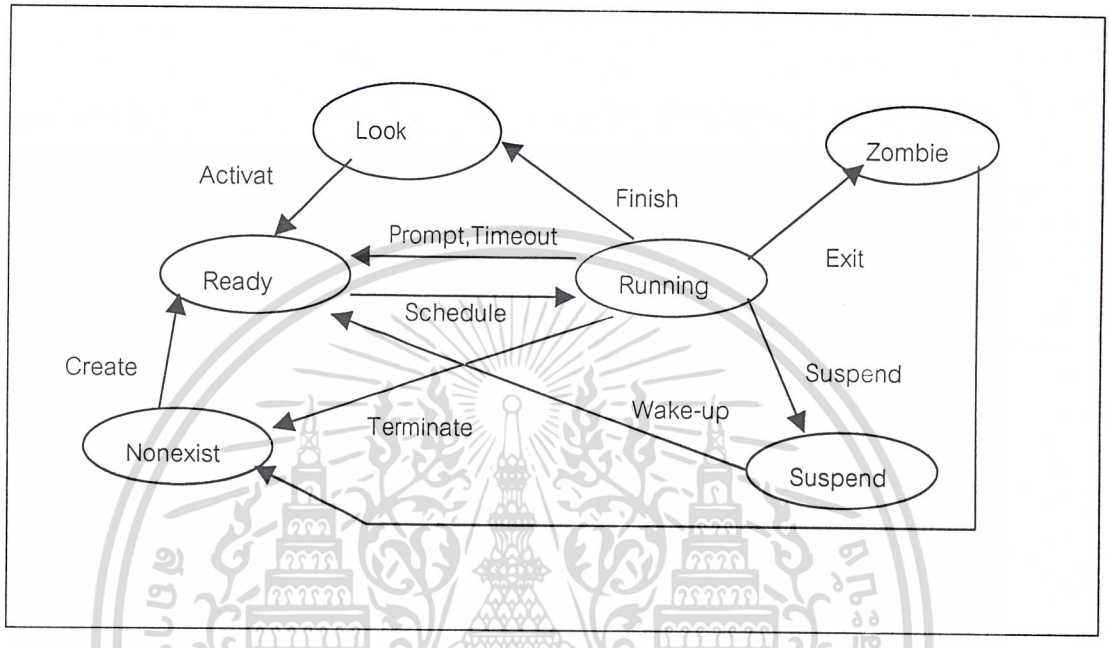
```
#pragma parallel
#pragma shared ( A, b, c )
#pragma local ( i )
{
        #pragma pfor iterate ( i=0; N ; 1)
                for ( i = 0 ; i < N ; i++) A[i] = b[i] * b[i+1] ;
        #pragma synchronize
        #pragma pfor iterate (i=0; N ; 1)
                for ( i = 0 ; i < N ; i++) c[i] = A[i] + A[i+1] ;
}
```

(d) Equivalent code using pragmas in SGI Power C

รูปที่ 3-3 ตัวอย่างโค้ดของการโปรแกรมเชิงขนานทั้งสามแบบ

3.2 โปรเซส

คำนิยาม โปรเซสคือส่วนหนึ่งของการรันโปรแกรมในขณะเวลานั้น ในโปรแกรมหนึ่งๆสามารถแบ่งงานออกได้เป็นหลายๆงานย่อย (โดยเฉพาะโปรแกรมเชิงขนาน) โดยแต่ละงานย่อยเราจะเรียกว่าโปรเซส และโปรเซสต่างๆเหล่านี้สามารถรันได้พร้อมๆกันด้วย รูปที่ 3.4 แสดงสถานะของการประมวลโปรเซส



รูปที่ 3-4 แผนภาพการเปลี่ยนแปลงสถานะของโปรเซส

- เมื่อเริ่มต้นยังไม่มีโปรเซสใดๆเกิดขึ้น จะเข้าสู่สถานะ "**nonexistent status**" จุดนี้โปรเซสจะถูกสร้างขึ้นโดยโปรเซสแม่จะกำเนิดโปรเซสลูกขึ้นมาเพื่อพร้อมประมวลผลงานในทันที
- "**suspended**" คือสถานะที่โปรเซสไม่สามารถทำงานต่อไปได้แล้วเนื่องจากเกิดความผิดพลาดขึ้น เช่น เพจฟอลต์ (page fault) เป็นต้น
- เมื่อสถานะต่างๆเปลี่ยนไปจากเดิม โปรเซสอาจจะสามารถทำงานต่อไปได้โดยการปลุก (wake up) เข้าสู่สถานะพร้อมทำงาน (ready)
- ถ้ามีหลายๆโปรเซสทำงานพร้อมกัน ซีพียูจะแบ่งเวลาไปให้แต่ละดปรเซสเท่าๆกัน เมื่อหมดเวลาของโปรเซสใดแล้ว โปรเซสนั้นจะเกิดการ ไทม์เอาต์ (time out) และจะกลับเข้าสู่สถานะพร้อมทำงานเพื่อให้โปรเซสอื่นๆได้ทำงานต่อไป
- สุดท้าย โปรเซสสิ้นสุดการทำงานทุกอย่างแล้วจะเข้าสู่สถานะ "**Exit**"

3.3 ขนาดพื้นที่ใช้งาน

ขนาดของพื้นที่ใช้งานหมายถึงมี2แบบคือ พื้นที่เก็บข้อมูลหรือโค้ดที่ใช้ในการประมวลผล เรียกว่าเวอร์ชวลแอดเดรส (virtual address) และพื้นที่ที่ซีพียูสามารถอ้างอิงถึงได้ทางกายภาพเรียกว่า ฟิสิกคอลแอดเดรส (Physical address) ขนาดของพื้นที่ใช้งานของซีพียูที่นิยมแพร่หลาย แสดงดังในตารางที่ 3.2

Architecture	Model	Size of Physical Address Space	Size of Virtual Address Space
Intel 80x86	8086	1 MB	1 MB
	Pentium	4 GB	64 TB
Power PC	601	4 GB	4 PB (2^{52} B)
	620	16 EB (2^{64} B)	16 EB
DEC Alpha	21064	16 GB	8 TB
	21164	1 TB	8 TB
MIPS	R4000	64 GB	1 TB
	R10000	1 TB	16 TB

ตารางที่ 3-2 ขนาดของพื้นที่ใช้งานของซีพียูแบบต่างๆ

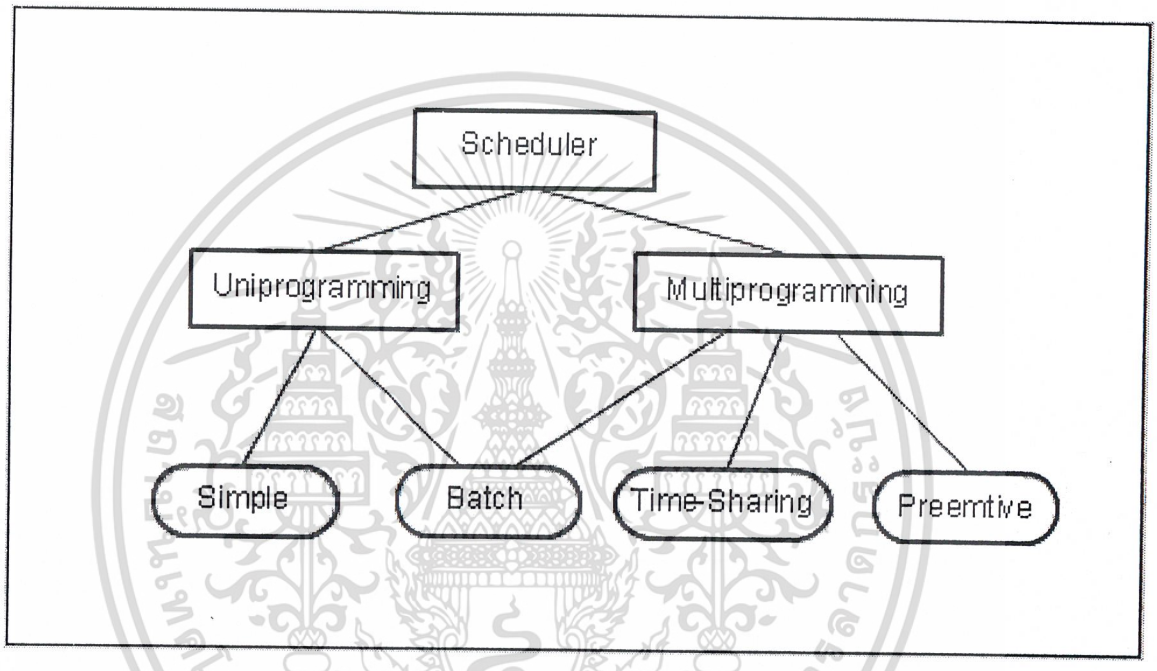
จากตารางที่ 3.2 "Intel Pentium Processor" มีการอ้างอิงตำแหน่งข้อมูลทางกายภาพขนาด 32 บิท ทำให้มีขนาดพื้นที่ใช้งานได้ถึง 4 GB ในโปรเซสเซอร์แบบ "DEC Alpha 21064" มีการอ้างอิงตำแหน่งข้อมูลถึงขนาด 16 GB (34 บิท) แต่ในเครื่อง Cray ที่มีการใช้โปรเซสเซอร์ชนิดนี้ จะอ้างอิงข้อมูลเพียง 64 MB เท่านั้น เพราะพื้นที่ใช้งานเหล่านี้จะถูกใช้ร่วมกับโปรเซสเซอร์ตัวอื่นๆ ในเครื่อง T3D ทีมีการใช้โปรเซสเซอร์ถึง 2048 ตัว ทำให้สามารถอ้างพื้นที่ใช้งานได้ถึง 2048 x 64 MB หรือ 128 GB เป็นต้น

3.4 การควบคุมโปรเซส

การควบคุมโปรเซสของเคอร์เนลประกอบด้วย การสร้างโปรเซส, การหยุดการทำงานของโปรเซส, การปลุกโปรเซสขึ้นมาทำงาน และจัดการโครงสร้างข้อมูลทั้งหมด เช่น การจัดลำดับการทำงานก่อนหลังของโปรเซส, สถานะของโปรเซส และรายการโปรเซสทั้งหมด เป็นต้น ซึ่งเคอร์เนลจะทำงานดังไปนี้

3.4.1 การป้องกัน (protection) เป็นการป้องกันของเคอร์เนลจากโปรเซสต่างๆ ไม่ให้เข้าถึงข้อมูลที่โปรเซสนั้นไม่มีสิทธิ์เช่น ทรัพากรของระบบ และข้อมูลข้องโปรเซสอื่น โปรเซสใดๆจะมีสิทธิ์เข้าถึงเฉพาะข้อมูลของตัวเองเท่านั้น เช่นในยูนิกซ์ จะมีกลไกไอพีซี (IPC Interprocess Communication) ทำการส่งข้อความเพื่อติดต่อกันระหว่างสองโปรเซสใดๆในระบบ เป็นต้น

3.4.2 การจัดตารางงาน (sheduling) เคอร์เนลจะทำการแบ่งปันทรัพยากรระบบต่างๆเช่น หน่วยความจำ, อุปกรณ์รอบข้าง และซีพียู ให้กับโปรเซสต่างๆได้รันงานอย่างเท่าเทียมกัน แต่ก็มีหลักการในการจัดแบ่งอยู่คือดูจากลำดับความสำคัญ โปรเซสที่มีลำดับความสำคัญสูงกว่าควรจะได้ประมวลผลก่อนหรือประมวลผลนานกว่าหรือได้ทรัพยากรระบบมากกว่าโปรเซสที่มีลำดับความสำคัญต่ำ เป็นต้น และเมื่อเปรียบเทียบลำดับความสำคัญที่เท่ากันของโปรเซสแล้ว เคอร์เนลจะต้องทำการแบ่งทรัพยากรของระบบให้กับโปรเซสอย่างเท่าเทียมและยุติธรรมด้วย คือไม่มีโปรเซสในได้เวลาในการประมวลผลนานกว่าโปรเซสอื่น หรือโปรเซสที่มาก่อนควรจะได้ประมวลผลก่อนเป็นต้น โดยรูปที่ 3.5 จะแสดงชนิดของการจัดตารางงาน



รูปที่ 3-5 ชนิดของการจัดตางรางงาน

- **แบช (batch)** คือการจัดตารางงานโดยการจัดเตรียมไว้ก่อนเพียงครั้งเดียว แล้วจึงทำการประมวลผลตามตารางนั้นๆไปจนจบ จะไม่มีการแทรกซ้อนจากภายนอกใดๆทั้งสิ้น มีข้อเสียคือโปรเซสอาจต้องเสียเวลาในการติดต่อกับอุปกรณ์ภายนอกซึ่งใช้เวลานานกว่ามากเมื่อเทียบกับการประมวลผลคำสั่งจากหน่วยความจำ
- **ไทม์แชริ่ง (time sharing)** เป็นการแบ่งปันเวลาในการประมวลผลหลายๆโปรเซสพร้อมๆกัน โดยเคอร์เนลจำทำการแบ่งเวลาให้กับโปรเซสคนละส่วนเรียกว่า ไทม์ควอนตัม (time quantum) เมื่อหมดไทม์ควอนตัมของโปรเซสแล้ว โปรเซสนั้นจะถูกหยุดรอแล้วจึงทำการนำโปรเซสใหม่ขึ้นมาทำงานแทนที่โปรเซสเดิม จนกระทั่งหมดไทม์ควอนตัมของโปรเซสใหม่นั้น วนเวียนเช่นนี้ไปเรื่อยๆ

- **พรีมทีฟ ซิสเต็ม (preemtive system)** จะดูความสำคัญของโปรเซสเป็นหลัก คือถ้าโปรเซสให้มีความสำคัญมากกว่า ก็จะสามารถหยุดการทำงานของโปรเซสที่กำลังรันอยู่ในขณะนั้น และนำตัวเองขึ้นไปรันแทนที่ได้

3.5 รูปแบบของโปรเซสแบบขนาน

ส่วนประกอบของโปรเซสในโปรแกรมเชิงขนานสามารถแบ่งออกได้เป็นชนิดใหญ่ๆได้ 3 ชนิดคือ

- **SPMD** single-program-multiple-data หมายถึงโค้ดตัวเดียวกันนำไปรันบนหลายๆโปรเซส แต่ใช้ข้อมูลในแต่ละโปรเซสต่างกัน
- **MPMD** multiple-program-multiple-data หมายถึงโปรเซสต่างๆรันงานจากโค้ดคนละส่วนกัน
- **SIMD** มีความละเอียดกว่า SPMD คือจะทำการประมวลผลคำสั่งเดียวกัน ณ เวลาเดียวกันโดยแต่ละโปรเซสใช้ข้อมูลต่างกัน

3.5.1 พาราเลลบล๊อค

วิธีการในการกำหนดของเขตของการแตกโปรแกรมออกเป็นโปรเซสย่อยจะเรียกว่าโครงสร้างพาราเลลบล๊อค (parallel block cinstructs) สามารถกำหนดได้ดังนี้

parbegin S_1, S_2 ... S_n **parend**

โดย S_1 S_2 ... S_n เป็นองค์ประกอบของโปรเซสนั้นๆ โดยเมื่อเริ่มการประมวลผล จะแตกออกเป็นโปรเซสย่อยๆ n โปรเซสพร้อมๆกัน โดยแต่ละโปรเซสก็จะได้องค์ประกอบ S_1 S_2 ... S_n ตามลำดับไม่เหมือนกันและไม่ขึ้นต่อกันด้วย พาราเลลบล๊อคนี้จะสิ้นสุดเมื่อทุกๆโปรเซสจบการทำงานแล้ว

3.5.2 โครงสร้างของข้อมูลขนาน (Data Parallel Constructs)

ในการประมวลผลข้อมูลแบบขนานเช่นใน Fortran 90 หรือ HPF จะสามารถกำหนดโครงสร้างของข้อมูลขนานได้ เช่น

parfor (i := 1; i < N; i++) {C[i] = A[i]+B[i];}

ผู้ใช้งานสารถกำหนดให้ C = A + B ได้ดังต่อไปนี้

forall (i = 1, N) C[i] = A[i]+B[i]

3.6 การซิงโครไนซ์

การซิงโครในซ์คือการควบคุมโปรเซสเพื่อที่จะหยุดรอโปรเซสอื่นๆหรือประมวลผลต่อไป ซึ่งมีรูปแบบต่างๆดังนี้

- **Atomicity** โปรเซสหลายๆโปรเซสมักจะต้องขั้นชุดคำสั่งช่วงหนึ่งเสมือนเป็นคำสั่งเพียงคำสั่งเดียวซึ่งเราเรียกว่าอะตอมมิก ซึ่งการประมวลผลอะตอมมิกจะต้องเสร็จสิ้นทั้งชุดคำสั่งไม่มีการขัดจังหวะ จึงจะถือว่าเสร็จสมบูรณ์ เช่น

```
parfor ( i := 1; i < n; i++){
        atomic {X = X+1; y=y-1;}
}
```

- **Control Synchronization** เป็นการควบคุมการประมวลผลของโปรเซสให้หยุดรอโปรเซสหนึ่งเมื่อถึงคำสั่งควบคุมที่เกิดขึ้น เพื่อประโยชน์ของการโปรแกรมแบบขนานบางอย่าง ยกตัวอย่างเช่นคำสั่ง "barrier" ดังต่อไปนี้

```
parfor (i:=1; i<n; i++){
        Pi
        barrier
        Qi
}
```

เมื่อโปรเซสคำนวณมาถึง Pi แล้วจะเข้าสู่คำสั่ง barrier เพื่อทำการหยุดรอโปรเซสอื่นๆ จนกระทั่งทุกโปรเซสทำการคำนวณมาถึงคำสั่ง barrier แล้วจึงจะทำการประมวลผลต่อไป อีกคำสั่งหนึ่งซึ่งมีการทำงานคล้ายๆกันคือคำสั่ง critical ดังตัวอย่างต่อไปนี้

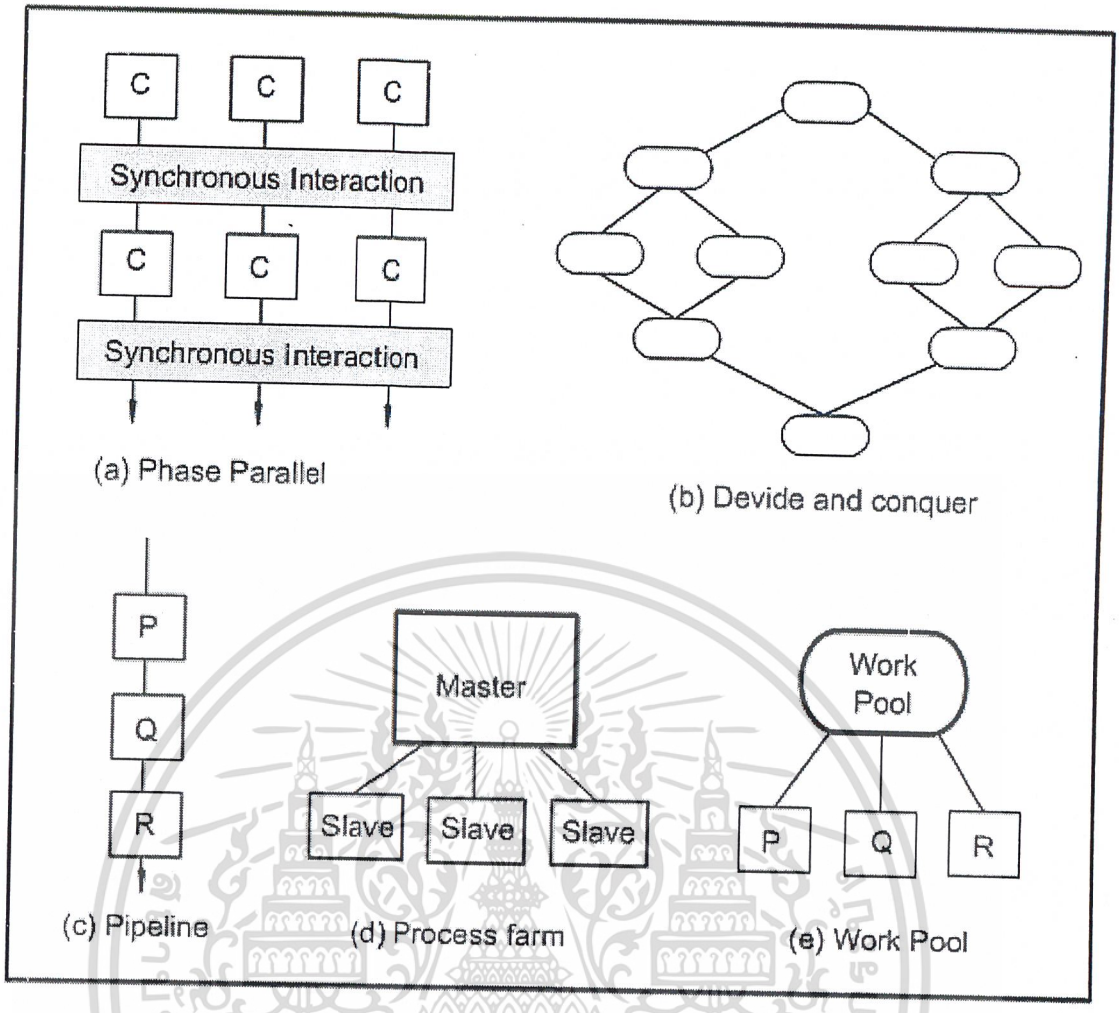
```
parfor (i:=1; i<n; i++){
        critical {X = X +1; y=y-1}
}
```

- **Data Synchronization** เมื่อโปรเซสต้องการจะเข้าถึงข้อมูล ซึ่งข้อมูลนั้นอาจจะใช้ร่วมกับข้อมูลอื่นๆ คำสั่งที่ใช้ควบคุมการเข้าถึงข้อมูลนั้น เช่น **wait (X>0)** เป็นการหยุดรอข้อมูลจนกระทั่ง X มีค่าเป็นบวกและยังมีคำสั่งต่างๆอีกเช่น looks, monitors และ events เป็นต้น ซึ่งเป็นการควบคุมข้อมูลเมื่อมีเงื่อนไขเกิดขึ้นดังตัวอย่างต่อไปนี้

```
parfor (i:=1; i<n; I++){look(S); X=X+1;y=y-1;unlock(S);}
```

3.7 อัลกอริทึมของโปรแกรมขนาน

ในการโปรแกรมเชิงขนานนั้นมีทางเลือกมากมายในการเลือกใช้อัลกอริทึมต่างๆ ซึ่งสรุปได้ดังรูปที่ 3.6

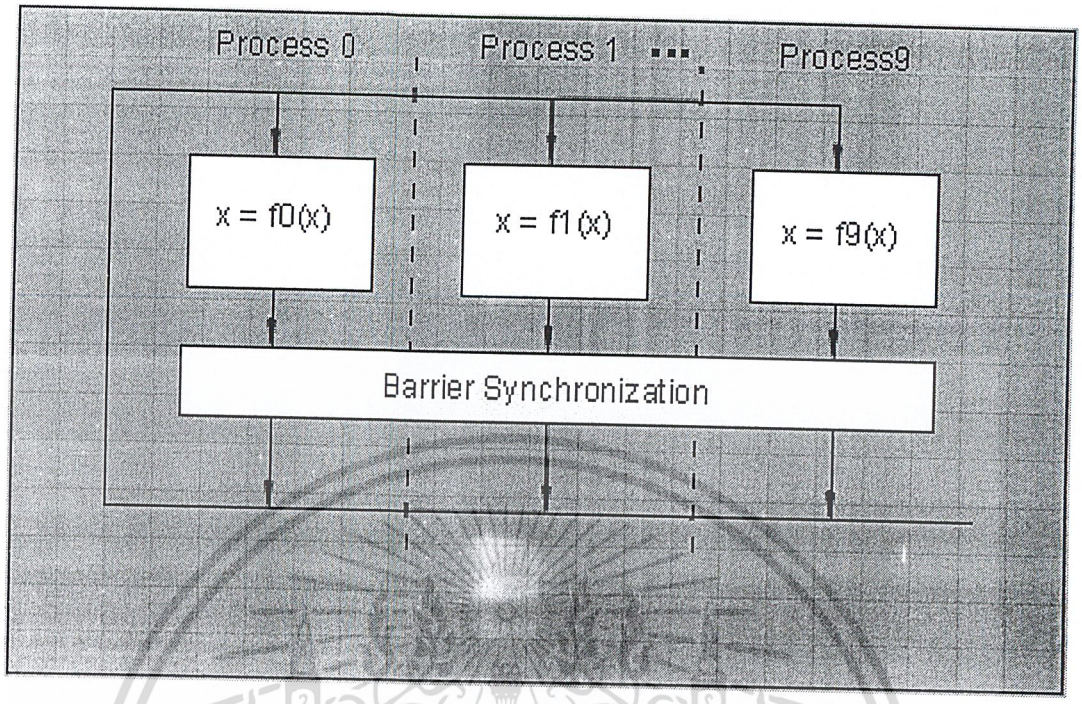


รูปที่ 3-6 อัลกอริทึมทั้ง 5 แบบของการโปรแกรมเชิงขนาน

3.7.1 เฟสพาราเลล (Phase Parallel)

เป็นวิธีที่นิยมแพร่หลายในการเขียนโปรแกรมเชิงขนานทั่วๆไป ดังในรูปมีโปรเซสหลายๆโปรเซส แต่ละโปรเซสจะไม่ขึ้นต่อกัน โดยมีการกระทำซิงโครไนซ์ barrier หรือการบล๊อคเป็นต้น ความยากของอัลกอริทึมแบบนี้คือการทำให้งานในแต่ละโปรเซสสมดุลกันตามความเหมาะสม ในรูปส่วนที่เป็นซิงโครไนเซชัน เป็นการทำให้เกิดจังหวะความลงตัวของทุกโปรเซส ซึ่งสามารถยกตัวอย่างเป็นโปรแกรมได้ดังนี้

```
parfor (i=0; i<N;i++) {
{
        for (j=0; j<N; j++){
                X[i] = F_i(X);
                barrier;
        }
}
```



รูปที่ 3-7 การซิงโครไนซ์ด้วยคำสั่ง barrier

จากโค้ดข้างต้นเป็นการหา X = f(X) โดยโปรแกรมจะสามารถกระจายออกเป็นโปรเซสย่อยๆได้ n โปรเซส โดยที่แต่ละโปรเซสเป็นอิสระต่อกันและถูกซิงโครไนซ์โดยคำสั่ง barrier เพื่อหยุดรอให้ทุกโปรเซสทำงานได้ถึงจุด barrier ก่อน จึงจะทำงานต่อไป จากโปรแกรมข้างบนเมื่อทำ barrier แล้วจะสิ้นสุดโปรแกรมทันที

3.7.2 ดีไวด์และคองเคอร์ (Divide and conquer)

เป็นหลักการที่คล้ายๆกับการโปรแกรมเชิงลำดับทั่วๆไปคือโปรเซสแม่จะแตกงานออกเป็นโปรเซสลูกหลายๆโปรเซสและเมื่อโปรเซสลูกได้ผลลัพธ์จากการประมวลแล้วจะนำผลลัพธ์ที่ได้มายุบรวมกันจนเหลือแต่เพียงโปรเซสแม่ ทำซ้ำเช่นนี้ไปเรื่อยๆจนได้ผลลัพธ์สุดท้ายในที่สุด อัลกอริทึมแบบนี้เหมาะกับงานประเภทควิกซอร์ท (quick sort) เพราะมีหลักการคล้ายๆกัน ข้อเสียของอัลกอริทึมแบบนี้คือการทำให้งานในแต่ละโปรเซสสมดุลกันนั้นทำได้ยาก

3.7.3 ไปป์ไลน์ (Pipeline)

งานจะถูกนำเข้าสู่ไปป์ไลน์ดังรูปและมีการประมวลผลอย่างต่อเนื่องกัน ในขณะที่ไปป์ไลน์สามารถมีได้หลายอัน และสามารถทำงานได้พร้อมๆกันด้วย เพียงแต่อาจจะมีการเหลื่อมลำกันของงานในแต่ละโปรเซส

3.7.4 โปรเซสฟาร์ม (Process Farm)

มาสเตอร์ (Master) จะประมวลผลส่วนที่เป็นคำสั่งแบบลำดับและกระจายส่วนที่มาสารถทำขนานกันได้ออกไปให้สลาฟ (Slave) ทำงาน เมื่อสลาฟทำงานเสร็จมาสเตอร์จะจ่ายงานใหม่ให้ไป จนกระทั่งไม่มีโปรเซสใดๆรันอยู่และงานได้ประมวลผลจนหมดแล้วถือว่าเป็นการทำงานที่สมบูรณ์ ข้อเสียของวิธีแบบนี้คือ จะเกิดปัญหาคอขวดขึ้นตรงส่วนของมาสเตอร์

3.7.5 เวอร์คพูล (Work Pool)

มักถูกใช้ในแบบจำลองแบบตัวแปรร่วม (shared variable) โดยมีการทำงานดังนี้ พูล (pool) จะเป็นส่วนที่เก็บงานต่างๆ โปรเซสจะถูกสร้างขึ้นและจะนำงานจากพูลไปประมวลผล งานต่างๆอาจมีการเพิ่มเข้าไปในพูลได้ การทำงานจะสิ้นสุดเมื่อไม่มีงานใดๆเหลืออยู่ในพูลและโปรเซสทุกๆโปรเซสทำงานจบแล้ว

3.8 แบบจำลองของการโปรแกรมเชิงขนาน

3.8.1 แบบจำลองอิมพลิซิท (Implicit Model)

เป็นการสร้างโปรแกรมเชิงขนานโดยที่โปรแกรมเมอร์ไม่จำเป็นต้องกำหนดโคดเอง ไม่จำเป็นต้องใช้ภาษาพิเศษ, ไลบรารี หรือฟังก์ชันของตัวแปลภาษา เพียงแต่โปรแกรมเมอร์เขียนโค้ดที่เป็นแบบโปรแกรมเชิงลำดับทั่วๆไปเท่านั้น แล้วนำโปรแกรมที่ได้มาวิเคราะห์ส่วนที่ไม่ขึ้นต่อกันของโปรแกรมเชิงลำดับ และใช้ตัวแปลภาษาทำการแปลงภาษาเชิงลำดับนั้นไปเป็นโปรแกรมเชิงขนาน

3.8.2 แบบจำลองดาต้าพาราเลล (Data Parallel Model)

เป็นการประมวลผลคำสั่งตัวเดียวกันแต่ใช้ข้อมูลที่ต่างกันในขณะเวลาเดียวกัน การประมวลแบบนี้เป็นเทรดเดี่ยว (single thread) แต่มีการขนานกันในกลุ่มของข้อมูล โดยจะมีการกำหนดชื่อของข้อมูลแบบใช้อ้างอิงได้ร่วมกัน และมีการซิงโครไนซ์กันอย่างหลวมๆ ยกตัวอย่างโค้ดการคำนวณหาค่า π ดังนี้

```
main() {
        double local[N], tmp[N],pi,w ;
        long i,j,t, N=1000000 ;
A:      w = 1.0 / N;
B:      forall (i =0;i<N;i++){
P:      local[i]=(i - 0.5 ) * w ;
Q:      tmp[i] = 4.0 / ( 1.0 + local[i] * local[i] ) ;
        }
C:      pi  = sum(tmp) ;
D:      printf("pi is  %f\n",pi*w) ;
} /* main() */
```

จะเห็นว่าสเตทเมนท์ P สามารถประมวลผลได้พร้อมๆกัน N สมการซึ่งเป็นการใช้ข้อมูลที่ประมวลผลขนานกันได้

3.8.3 แบบจำลองเมสเซจพาสซิ่ง (Message Passing Model)

แบบจำลองเมสเซจพาสซิ่งนี้มีหลักการคล้ายๆกับการส่งข้อความติดต่อสื่อการกันของโปรเซสโดยผ่านตัวกลางรับส่งของระบบ ซึ่งเป็นที่นิยมใช้งานในการโปรแกรมเชิงขนานเช่น PVM และ MPI และใช้กับคลัสเตอร์ในวิทยานิพนธ์เล่มนี้ด้วย แบบจำลองเมสเซจพาสซิ่งมีคุณลักษณะดังต่อไปนี้

- **Asynchronous Parallelism** การซิงโครไนซ์ของโปรเซสใช้คำสั่งอย่างเช่น barrier เป็นตัวจัดการ
- **Separate Address Space** แต่ละโปรเซสจะมีการอ้างถึงพื้นที่ใช้งานต่างกันและไม่สามารถมองเห็นได้จากโปรเซสอื่นด้วย การถ่ายข้อมูลระหว่างตัวแปรของโปรเซสจะใช้การส่งเมสเซจเป็นหลัก
- **Explicit Interaction** การสร้างโค้ดต่างๆ, การกระทำของโปรเซส, การซิงโครไนซ์และการโหลดงาน โปรแกรมเมอร์ต้องกำหนดกฎเกณฑ์ขึ้นมาเองทุกอย่าง
- **Explicit Allocation** การจองงานและข้อมูลสำหรับโปรเซสนั้นกระทำโดยโปรแกรมเมอร์

ตัวอย่างโค้ดที่เขียนแบบเมสเซจพาสซิ่ง ในการคำนวณหาค่า π

```
#define  N          1000000
         main() {
                  double local, pi, w;
                  long i, taskid, numtask;
A:       w = 1.0 / M;
         MPI_Init( &argc, &argv );
         MPI_Comm_rank( MPI_COMM_WORLD, &taskid);
B:       for ( i = taskid; i<N; i=i+numtask) {
                  local = ( i + 0.5 ) * w ;
                  local = 4.0 / ( 1.0 + local * local);
         }
         MPI_Reduce ( &local, &pi,1,MPI_Double,
                  MPI_MAX,0,MPI_COMM_WORLD);
         if(taskid == 0) printf("pi is  %f\n",pi*w) ;
         MPI_Finalize();
} /* main() */
```

3.8.4 แบบจำลองแชร์วาริเอเบิ้ล (Shared Variable Model)

แบบแชร์วาริเอเบิ้ลมีความคล้ายคลึงกับแบบดาต้าพาราเลลในแง่ของการอ้างถึงข้อมูลร่วมกัน หมายความว่าตัวแปรสามารถอ้างอิงจากโปรเซสใดๆก็ได้โดยใช้ชื่อในการอ้างชื่อเดียวกัน และยังมีความคล้ายกับแบบเมสเซจพาสซึ่งตรงการทำมัลติเทรดและการซิงโครไนซ์

ตัวอย่างของโค้ดแบบแชร์วาริเอเบิ้ล สำหรับการคำนวณหาค่า π

```
#define  N          1000000
         main() {
                  double local, pi = 0.0  w;
                  long i ;
A:                w = 1.0 / N ;
B                 #pragma parallel
                  #pragma shared ( pi , w )
                  #pragma local ( i, local )
                  {
                           #pragma pfor iterate ( i = 0;N;1)
```

```
                    for ( i = 0; i < N; i++) {
                    local = ( i + 0.5 ) * w ;
                    local = 4.0 / (1.0 + local * local ) ;
                    }
                    #pragma critical
                            pi = pi + local ;
            }
            printf(:pi is   %f\n",pi*w);
} /*main()*/
```

3.9 ข้อดีข้อเสียของแบบจำลองโปรแกรมเชิงขนานแบบต่างๆ

Issues		Implicit	Data Parallel	Message Passing	Shared Variable
Platform-independent examples		Kap, Forge	Fortran 90, HPF	PVM, MPI	X3H5
Platform-dependent examples			CM C*	SP2, MPL, Paragon Nx	Cray Craft, SGI Power PC
Parallelism issues		good	normal	bad	normal
Allocation issues		good	normal	bad	normal
Interaction issues	Communication	good	normal	bad	normal
	Synchronization	good	good	normal	bad
	Aggregation	good	normal	good	bad
	Irregularity	good	normal	normal	normal
semantic issues	Termination	good	good	normal	bad
	Determinacy	good	good	normal	bad
	Correctness	good	normal	normal	bad
Program ability issues	Generality	bad	normal	normal	normal
	Patability	good	normal	normal	bad
	Structuredness	good	normal	bad	bad

ตารางที่ 3-3 เปรียบเทียบข้อดีข้อเสียของแบบจำลองการโปรแกรมเชิงขนานแบบต่างๆ

- **อิมพลิซิท** มีข้อดีมากมายคือสามารถนำโปรแกรมเชิงลำดับที่เขียนแล้วมาใช้ได้ทันที ไม่ต้องมีการสร้างใหม่ และโปรแกรมเมอร์ไม่จำเป็นต้องมีความรู้ในการเขียนโปรแกรมเชิงขนาน
- **ดาต้าพาราเลล** มีการใช้พื้นที่ใช้หน่วยความจำแบบเดี่ยว ซึ่งไม่จำเป็นต้องจองข้อมูล, มีการทำเป็นเทรดเดี่ยว ทำให้ลดปัญหาการเกิดล็อคดาวน์ (lockdown) ได้
- **เมสเซจพาสซิ่ง** โปรแกรมเมอร์ต้องกำหนดงานในแต่ละโปรเซส, จับจองพื้นที่ใช้งานและการทำซิงโครไนซ์เองทั้งหมด แต่ก็มีข้อดีคือมีความคล่องตัวของการเขียนโปรแกรม
- **แชร์วาริเอเบิ้ล** เป็นแบบจำลองที่สามารถนำไปใช้ได้กับทุกแพลตฟอร์ม เช่น PVP, SMP, DSM, MPP หรือแม้กระทั่งคลัสเตอร์ เป็นต้น แต่กระนั้นก็ดีการซิงโครไนซ์ทำให้เกิดโอเวอร์เฮดสูงและทำให้ทำงานช้ากว่าแบบเมสเซจพาสซิ่งด้วย แต่การโปรแกรมจะง่ายกว่าแบบเมสเซจพาสซิ่งมาก

บทที่ 4
การโปรแกรมแบบเมสเซจพาสซิ่ง
(Message Passing Programming)

4.1 เมสเซจพาสซิ่งไลบรารี (Message Passing Libraries)

ในปัจจุบันมีซอฟต์แวร์เมสเซจพาสซิ่งเกิดขึ้นมากมายและส่วนใหญ่ก็เป็นซอฟต์แวร์ที่แจกจ่ายให้ฟรีด้วยดังแสดงบางส่วนในตารางที่ 3.1

Name	Original Creator	Distinct Feature
CMMD	Thinking Machines	Use Active Message for low latency
Express	Parasoft	Collective communication and I/O
Fortran-M	Argonne National Lab	Modularity and Determinacy
MPI	PMI Forum	A widely adopted standard
Nx	Intel	Originate from the Intel hypercube MPPs
P4	Argonne National Lab	Integrate shared memory and message passing
PARMACS	ANL/GMD	Mainly used in Europe
PVM	Oak Ridge National Lab	A widely used, stand-alone system
UNIFY	Mississippi State	A system allowing both MPI and PVM calls
Zipcode	Livermore National Lab	Contributed to the context concept

ตารางที่ 4-1 ตัวอย่างของซอฟต์แวร์แบบเมสเซจพาสซิ่ง

ผู้ใช้งานส่วนมากมักจะเลือก PVM หรือ MPI ในการพัฒนาโปรแกรมแบบเมสเซจพาสซิ่ง ผู้สร้างฮาร์ดแวร์หลายรายก็ออกแบบระบบของตนที่สนับสนุนการทำงานของทั้ง MPI และ PVM ด้วย เนื่องจากข้อดีหลายๆประการของมันคือ มีความเป็นมาตรฐาน ทุกที่ใช้รูปแบบเดียวกันหมด

4.2 แบบจำลองเมสเซจพาสซิ่ง

เมสเซจพาสซิ่งเป็นการติดต่อสื่อการกันและการทำซิงโครไนซ์กันระหว่างโปรเซส โดยมีการกระทำหลักๆอยู่ 2 อย่างด้วยกันคือส่งและรับ ซึ่งสามารถใช้ได้ 3 ลักษณะดังจะได้อธิบายในส่วนต่อไป ซึ่งข้างล่างเป็นโค้ดตัวอย่างประกอบ

ตัวอย่างการส่งและรับเมสเซจ

```
        process P:                  process Q:
        M = 10;                     s = - 100
L1:     send M to Q ;       L1:     receive S from P ;
L2:     M = 20 ;            L2:     X = S + 1 ;
        goto L1 ;
```

จากตัวอย่างข้างต้น โปรเซส P ส่งเมสเซจที่ประกอบด้วยตัวแปร M ไปยังโปรเซส Q ซึ่งรับตัวแปรเข้ามาไว้ใน S

รูปแบบการส่งและรับเมสเซจ

- **Synchronous Message Passing** เมื่อโปรเซส P คำนวณ "synchronous send M to Q" แล้ว จะรอจนกระทั่งโปรเซส Q คำนวณ "synchronous receive S from X" คือโปรเซสทั้งคู่จะไม่เสร็จสิ้นการทำงานจนกว่า M จะถูกส่งออกไป และ S ถูกรับเข้ามาแล้วทั้งคู่ ในกรณีนี้ X ควรจะมีค่าเท่ากับ 11
- Blocking Send/Receive การบล๊อคการส่ง blocking send จะทำให้โปรเซสหยุดรอจนกระทั่งเมสเซจถูกส่งออกไปแต่ทางฝั่งรับไม่จำเป็นต้องรับหรือยังไม่รับในเวลานั้นก็ได้ จากนั้นโปรเซสจะทำงานต่อไปโดยไม่สนใจฝั่งรับ การส่งแบบนี้รับประกันได้เพียงว่า ข้อมูลที่ถูกส่งออกไปไม่ถูกเขียนทับก่อนจะถูกส่ง แต่ไม่รับประกันทางฝั่งรับว่าข้อมูลนั้นอาจจะยังไม่ได้รับแล้วมีข้อมูลใหม่เข้ามาทับข้อมูลเดิมทำให้ข้อมูลเก่าหายไป

  การบล๊อคการรับคือ โปรเซสหยุดรอเมสเซจจนกระทั่งได้รับเมสเซจนั้นเรียบร้อยแล้วถึงจะทำงานส่วนอื่นของโปรเซสต่อไป และจากโค้ดการทำงานข้างต้น เมื่อใช้ "blocking send/receive" แล้ว X จะมีค่าเท่ากับ 11
- **nonBlocking Send/Receive** เป็นการรับส่งโดยที่ไม่สนใจว่าข้อมูลนั้นจะถูกส่งออกไป หรือรับเข้ามาหรือไม่ เมื่อทำคำสั่งส่งหรือรับแล้ว จะกลับไปทำงานส่วนอื่นของโปรเซสต่อไปทันทีไม่มีการหยุดรอ โดยข้อมูลที่กำลังจะส่งหรือรับนั้นอาจยังไม่ถูกส่งออกไปในเวลานั้น ซึ่งก็อาจจะถูกเขียนทับก่อนถูกส่งออกไปก็ได้ ในกรณีนี้ค่าของ X อาจเป็น 11 21 หรือ –99 ก็ได้ ดังนั้นเราควรเขียนโค้ดข้างบนเพื่อให้เกิดความถูกต้องของข้อมูลโดยการใช้การหยุดรอการรับส่งดังนี้

```
        process P:                  process Q:
        M = 10 ;                    S = - 100
L:      send M to Q ;               receive S from P
        wait for M to be sent       wait for s to be received
        M = 20 ;                    X = S + 1 ;
```

4.3 เอ็มพีไอ (MPI Message Passing Interface)

เอ็มพีไอเป็นมาตรฐานของเมสเซจพาสซิ่งแบบหนึ่งที่นิยมใช้งานแพร่หลายและมีการพัฒนาให้ใช้งานได้หลายแพลตฟอร์มหลายภาษาเช่น Fortran, C เป็นต้น เอ็มพีไอมีประสิทธิภาพในการเขียนโปรแกรมเชิงขนานสูงเนื่องจากมีฟังก์ชันให้ใช้งานกว่า 200 ฟังก์ชัน และการเรียนรู้ก็ง่ายกว่าพีวีเอ็ม (มีอธิบายในส่วนต่อไป) ด้วย

```
#include "mpi.h"
int foo(i) int i ; {..}
main(argc, argv)
int argc;
char *argv[] ;
{       int i, tmp, sum =0, group_size, my_rank, N;
        MPI_Init (&argc, &argv);
        MPI_Comm_size (MPI_COMM_WORLD, &group_size);
        MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
        if (my_rank = =0){
                printf("Enter N:");scanf("%d",&N);
                for (i=1; i<group_size; i++)
S1:                     MPI_Send(&N,1,MPI_INIT,i,i,MPI_COMM_WORLD);
                for (i=my_rank ;i<N ; i = i + group_size )
                        sum = sum + foo(i) ;
                for (i=1; i<group_size; i++) {
S2:                     MPI_Recv(&tmp,1,MPI_INIT,i,i,MPI_COMM_WORLD,&status) ;
                        sum = sum + tmp ;
                }
                printf ("\n The resul = %d",sum ) ;
        }
        else {  /* if my_rank != 0 */
S3:             MPI_Recv(&N,1,MPI_INIT,0,i,MPI_COMM_WORLD,&status) ;
                for ( i = my_rank ; i<N ; i = i + group_size )
                        sum= sum + foo(i) ;
                MPI_Send(&sum,1,MPI_INIT,0,i,MPI_COMM_WORLD);
        }
        MPI_Finalize() ;
}
```

รูปที่ 4-1 โปรแกรมที่เขียนโดยใช้เอ็มพีไอ

จากโปรแกรมข้างต้นเริ่มด้วยการกำหนดค่าให้กับโปรเซสด้วย

MPI_COMM_size และ

MPI_COMM_rank

และทำการติดต่อสื่อสารกันด้วยการใช้คำสั่ง

MPI_send และ

MPI_recv

และสุดท้ายเมื่อเสร็จสิ้นกระบวนการทุกอย่างแล้วจึงทำการเรียก

MPI_Finalize

เพื่อทำการจบการทำงานโดยสมบูรณ์
สมมุติโปรแกรมตัวอย่างในรูปที่ 4.1 ชื่อว่า "myprog.o"
ในการคอมไพล์โดยใช้คอมไพเลอร์ mpcc จะทำดังนี้

mpcc myprog.c –o myprog

และการรันงานที่คอมไพล์แล้วดังคำสั่งต่อไปนี้

MPIRUN myprog –np -n

โดย n คือจำนวนโปรเซสที่เราต้องการ

นอกจากคำสั่งที่กล่าวมาข้างต้นแล้ว เอ็มพีไอยังต้องการโปรแกรมจัดการสภาพแวดล้อมของโปรเซสเพื่อรันงานด้วย โดยเอ็มพีไอจะอยู่ชั้นบนสุด, โปรแกรมจัดการสภาพแวดล้อมอยู่ตรงกลาง และมีระบบปฏิบัติการอยู่ล่างสุด ยกตัวอย่างโปรแกรมจัดการสภาพแวดล้อมเช่น LAM (Local Area Multicomputer) ใช้สำหรับจัดการสภาพแวดล้อมบนยูนิกซ์คลัสเตอร์ ซึ่งจะมีอธิบายแหล่งที่มาและวิธีใช้ในบทต่อไปด้วย

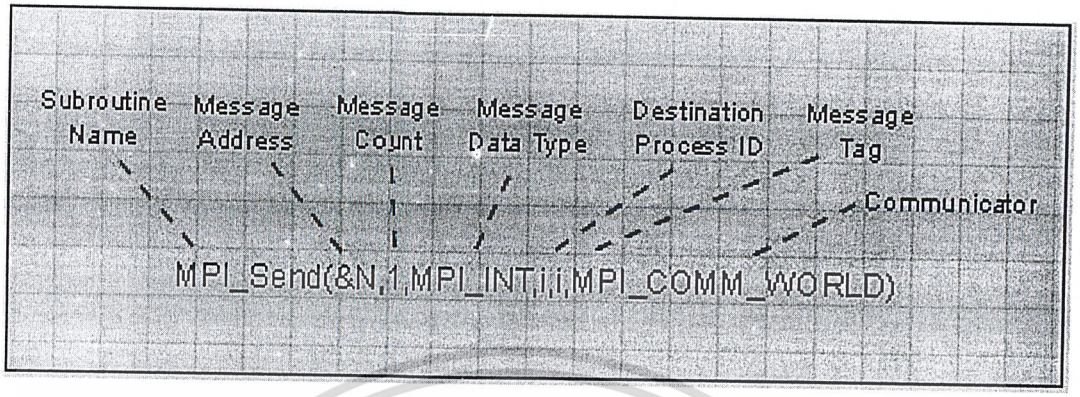
4.3.1 เมสเซจ

เมสเซจคือข้อความคล้ายกับจดหมายที่ระบุผู้ส่งผู้รับและมีเนื้อความของจดหมาย ดังตัวอย่างข้างล่างนี้

```
MPI_Send ( &N,1,MPI_INT,i,i,MPI_COMM_WORLD) ;
MPI_Revc(&n,MPI_INT,0,i,MPI_COMM_WORLD,&status) ;
```

ซึ่งจะเปรียบเทียบได้กับตัวอย่างก่อนหน้าเป็น

Send M to Q ; and Reveive S from P ;



รูปที่ 4-2 ความหมายของแต่ละส่วนของคำสั่ง MPI_Send

4.3.2 ชนิดข้อมูลของเอ็มพีไอ

ในภาษาทั่วๆไปนั้นย่อมต้องมีการกำหนดชนิดของข้อมูลเสมอ ในเอ็มพีไอก็เช่นกัน การกำหนดชนิดของข้อมูลจะกำหนดอยู่ในส่วนของการรับส่งเมสเซจ เนื่องจากว่าข้อมูลแต่ละชนิดนั้นมีความยาวของข้อมุลที่แตกต่างกัน จึงจำเป็นต้องกำหนดที่ฝ่ายส่งเพื่อให้ฝ่ายรับได้ทราบถึงชนิดของข้อมูลที่ถูกส่งออกมาเพื่อจะได้รับได้ถูกต้อง ชนิดของข้อมูลนั้นมีหลายๆแบบคล้ายๆกับภาษา C เช่น ไบต์, อักขระ, จำนวนเต็ม, ทศนิยม เป็นต้น ต่อไปเป็นตัวอย่างของการรับส่งของข้อมูลแบบอักขระ

MPI_Send(string,1001MPI_CHAR,China,…)

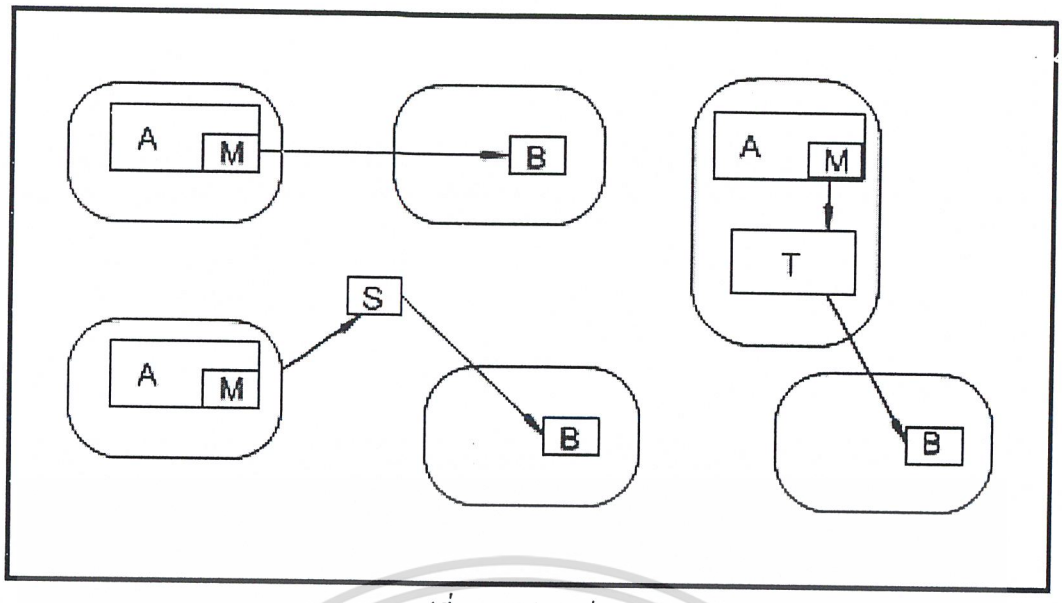
MPI_Recv(string,1001MPI_CHAR,USA,…)

100 หมายถึงจำนวนของข้อมูลโดย MPI_CHAR จะบ่งบอกว่าข้อมูลที่ส่งออกมานั้นเป็นแบบอักขระ

4.3.3 เมสเซจบัฟเฟอร์ (Message Buffer)

เมสเซจบัฟเฟอร์หรือเรียกย่อๆว่าบัฟเฟอร์คือพื้นที่หน่วยความจำสำหรับเก็บข้อมูลชั่วคราว ปกติบัฟเฟอร์จะถูกจองหรือสร้างขึ้นได้ใน 3 ลักษณะ

- พื้นที่หน่วยความจำที่โปรแกรมเมอร์จองไว้เองสำหรับเก็บข้อมูลที่ทำการรับส่งกัน
- ส่วนของหน่วยความจำที่ระบบสร้างขึ้นอัตโนมัติ สำหรับเก็บข้อมูลที่รับส่งกันไว้ชั่วคราว เมื่อการรับส่งเสร็จสิ้นแล้วบัฟเฟอร์ก็จะหายไปด้วย บัฟเฟอร์ชนิดนี้ผู้ใช้งานจะมองไม่เห็นและไม่สามารถเรียกใช้งานได้ เราเรียกบัฟเฟอร์ชนิดนี้ว่าบัฟเฟอร์ระบบ (system buffer)
- เอ็มพีไอ อนุญาติให้ผู้ใช้งานสามารถสร้างบัฟเฟอร์ขึ้นเองได้ บัฟเฟอร์แบบที่ 3 นี้จะจับจองพื้นที่สำหรับเก็บข้อมูลเมสเซจต่างๆที่ปรากฏในโปรแกรมดังรูปที่ 4.3



รูปที่ 4-3 บัฟเฟอร์แบบต่างๆ

4.3.4 เมสเซจแท็ก (message tag)

แทกคือส่วนที่ติดไปกับเมสเซจสำหรับบ่งบอกป้ายชื่อของเมสเซจนั้น ยกตัวอย่างเช่น

```
process P:                    process Q:
send (A, 32, Q, tag1)         recv ( X, P, tag1, 32)
send (B, 16, Q, tag2)         recv ( Y, P, tag2, 16)
```

จากโค้ดข้างบน tag1 และ tag2 จะถูกส่งไปกับเมสเซจ ทำหน้าที่เป็นป้ายชื่อบอกผู้รับและผู้ส่งเพื่อให้การรับส่งเป็นไปอย่างถูกต้อง ผู้รับและผู้ส่งต้องมี tag เหมือนกันเท่านั้นถึงจะทำการรับส่งกันได้ จากโค้ดข้างบน A จะถูกส่งเข้าไป X ในขณะที่ B ถูกส่งไปเข้า Y, จะไม่เข้าไปที่ Y และ B จะไม่เข้าไปที่ X เพราะว่ามี tag ไม่ตรงกัน ที่เราต้องกำหนด tag เพราะอาจเกิดความผิดพลาดเช่นกรณีที่ A และ B ถูกส่งออกมาจากโปรเซส P แต่ถ้า B มาถึงก่อน A แล้ว B .ซึ่งมีขนาดข้อมูล 16 บิท เข้าไปที่ X ซึ่งมีขนาดข้อมูล 32 บิท จะทำให้ผิดพลาดทันที

4.3.5 คอมมิวนิเคเตอร์ (Communicator)

คอมมิวนิเคเตอร์คือกลุ่มของโปรเซสและซุปเปอร์แท็กที่สามารถทำการรับส่งข้อมูลกันระหว่างกลุ่มได้ ซุปเปอร์แทกจะคล้ายๆกับแท็กคือเป็นป้ายชื่อของกลุ่มโปรเซสนั้นๆ ผู้เขียนโปรแกรมมักจะให้โปรเซสที่อยู่ในกลุ่มเดียวกันเท่านั้นที่สามารถติดต่อกันได้ ดังตัวอย่างการใช้งานต่อไปนี้

```
MPI_Send (MSG1,count1,MPI_INT,1,tag1,COMM1);
MPI_Recv (MSG1,count1,MPI_INT,0,tag1,COMM1);
```

4.4 พีวีเอ็ม (PVM)

พีวีเอ็มคือซอฟต์แวร์ที่ออกแบบมาเพื่อเครือข่ายยูนิกซ์คอมพิวเตอร์สำหรับการทำเมสเซจพาสซิ่ง ชนิดหนึ่งที่มีความนิยมแพร่หลายและสามารถนำไปใช้กับแพลตฟอร์มได้เกือบทุกชนิดไม่ว่าจะเป็น SMP, PVP, MPP หรือแม้กระทั่งคลัสเตอร์ นอกจากนั้นพีวีเอ็มยังสามารถประยุกต์ใช้กับแพลตฟอร์มที่ไม่ใช่ยูนิกซ์อื่นๆได้อีกเช่น Windows NT Windows95 ทั้งยังมีภาษาที่สยับสนุนมากมายอาทิเช่น C,Fortran, JAVA เป็นต้น

ประวัติการพัฒนา พีวีเอ็มเริ่มมีขึ้นเมื่อปี ค.ศ. 1989 ที่ Oak Ridge National Laboratory โดยทีมพัฒนาของมหาวิทยาลัย และได้มีการพัฒนาต่อกันมาเป็นลำดับจนถึงปัจจุบันโดยมีเป้าหมายเพื่อให้เป็นเมสเซจพาสซิ่งไลบรารี่บนแพลตฟอร์มยูนิกซ์ ผู้ใช้งานสามารถสร้างเวอร์ชวลแมชชีน (Virtual Machine) ที่ประกอบจากการเชื่อมต่อของโหนดแบบยูนิกซ์และผู้ใช้งานสามารถสร้างโปรเซสขึ้นมารันบนเวอร์ชวลแมชชีนนั้นได้โดยมีฟังก์ชันที่ทำการติดต่อระหว่างโปรเซสเป็นไลบรารีไว้ให้ด้วย

4.4.1 เปรียบเทียบพีวีเอ็มกับเอ็มพีไอ

พีวีเอ็มเป็นระบบที่สามารถจัดการได้ด้วยตัวเองไม่จำเป็นต้องมีโปรแกรมจัดการสภาพแวดล้อมอย่างเอ็มพีไอ แต่เอ็มพีไอจะมีฟังก์ชันเมสเซจพาสซิ่งที่มีประสิทธิภาพมากกว่าและมีมาตรฐานที่แน่นอนชัดเจน ส่วนพีวีเอ็มนั้นไม่มีการกำหนดมาตรฐานออกมาอย่างตายตัว ทำให้เกิดการแยกกันพัฒนาและเกิดความแตกต่างกันในแต่ละเวอร์ชันอย่างมาก

4.4.2 โครงสร้างของพีวีเอ็ม

พีวีเอ็มจะประกอบด้วย พีวีเอ็มเดมอน (PVM daemon , pvmd) ซึ่งจะรันอยู่ในทุกๆโหนดของเวอร์ชวลแมชชีนและเรียกใช้ไลบรารี (libpvm3.a) เชื่อมต่อกับโปรแกรมใช้งานในการจัดการโปรเซส, เมสเซจพาสซิ่ง และการจัดการเวอร์ชวลแมชชีน

พีวีเอ็มสามารถหาซอฟต์แวร์สำหรับติดตั้งได้มากมายจากเว็บ โดยเมื่อติดตั้งเสร็จแล้วจะทำการเรียกใช้งานดังนี้

```
pvm host_file
```

คำสั่งนี้เป็นการเริ่มต้นการทำงานของ pvmd ในทุกๆโหนดของเวอร์ชวลแมชชีน ที่อยู่ในไฟล์ชื่อ “host_file” หลังจากนั้นจะเป็นการเข้าสู่คอนโซลของพีวีเอ็มซึ่งมีลักษณะดังนี้

```
pvm>
```

คอนโซลของพีวีเอ็มนั้นเป็นส่วนที่รอรับคำสั่งคล้ายๆคอมมานไลน์ เราสามารถใส่คำสั่งพีวีเอ็มได้ที่คอนโซลนี้ ต่อไปเป็นคำสั่งของพีวีเอ็มที่มีใช้บ่อยๆ

Command	Operation
pvm>add apple	Add the host “apple” to the virtual machine
pvm>delete apple	Delete the host “apple” from the virtual machine
pvm>conf	List the configuration of the virtual machine
pvm>spawn –count 4 app	start $ task to run “app” on the virtual machine
pvm>jobs	List jobs running in the virtual machine
pvm>halt	Kill all PVM process and shutdown PVM

ตารางที่ 4-2 คำสั่งหลักของพีวีเอ็ม

4.4.3 ไดนามิกคอนฟิกกูเรชัน

เวอร์ชวลแมชชีนสามารถปรับแต่งตัวเองได้ตลอดเวลาไม่มีการกำหนดถาวร โดยการเรียกใช้ไลบรารีเช่น pvm_addhosts, pvm_deletehosts เป็นต้น ดังตัวอย่างข้างล่างนี้

```
int     info, nhost=2, infos[2];
char    *hosts[] = { “apple”,”orange.usc.edu”}
info = pvm_addhosts( hosts, nhost, infos);
info = pvm_delhosts( hosts, nhost, infos);
```

4.4.4 การจัดการโปรเซส

ที่คอนโซลของพีวีเอ็มนั้นเราสามารถจัดการกับโปรเซสต่างๆได้เช่น

```
pvm>spawn –count 4 foo
```

หมายถึงสร้างโปรเซสมา 4 โปรเซสขึ้นมารันงานชื่อ “foo” ในเวอร์ชวลแมชชีน คำสั่งดังกล่าวนั้นไม่ได้กำหนดให้โหนด ไหนรันโปรเซสใด แต่เราสามารถกำหนดได้ดังนี้

```
pvm>spawn –(apple) foo
```

หมายความว่าให้รันโค้ดจาก “foo” ที่โหนดชื่อ “apple” ส่วนคำสั่งที่ใช้จัดการโปรเซสแสดงในตารางที่ 4.3

PVM Function Call	Meaning
tid=pvm_mytid();	Get the task ID of the calling task
tid=pvm_parent();	Get the task ID of the parent task
info=pvm_exit();	The calling task exits PVM
numt=pvm_spawn(...);	Spawn a PVM task
info=pvm_kill(tid);	Terminate a PVM task
tstat=pvm_pstat(tid);	Get the status of a PVM task
info=pvm_task)(...);	Get the information of all tasks
mstat=pvm_mstat(host);	Get the status of a host
info=pvm_config(...);	Get the configuration information of virtual machine

ตารางที่ 4-3 คำสั่งของพีวีเอ็มที่ใช้ในการจัดการโปรเซส

ตัวอย่างของโปรแกรมที่เขียนโดยใช้พีวีเอ็มเพื่อคำนวนหาค่า π แสดงได้ดังนี้

```
#define n16
#include "pvm3.h"
main (int argc, char ** argv)
{
        int mytid,  tids[n], me, i, N, rc, parent;
        double mypi, h, sum=0.0, x ;
        me = pvm_joingroup( "PI");
        if (me ==0) {
                pvm_spawn("pi", (char**)0, 0, "", n-1, tids);
                scanf("%d",&N);
                pvm_initsend( PvmDataRaw );
                pvm_pkint(&N,1,1);
                pvm_mcast(tids,n-1,5);
        } else {
                pvm_recv(parent, 5);
                pvm_upkint(&N, 1, 1);
        }
        pvm_barrier("PI", n);
        h = 1.0 / (double) N;
        for (i = me + 1; i <= N; i+= n) {
                x = h*((double)i – 0.5);
                sum += 4.0 / (1.0 + x*x);
        }
        mypi = h * sum;
        pvm_reduce( PvmSum, &mypi, 1, PVM_DOUBLE, 6, "PI", 0);
        if (me==0) printf("pi is approximately %.16f\n",mypi);
        pvm_lvgroup("PI");
        pvm_exit();
}
```

บทที่ 5
การออกแบบและสร้างคลัสเตอร์แบบเบียวูล์ฟ
(Beowulf Cluster System Design and Implement)

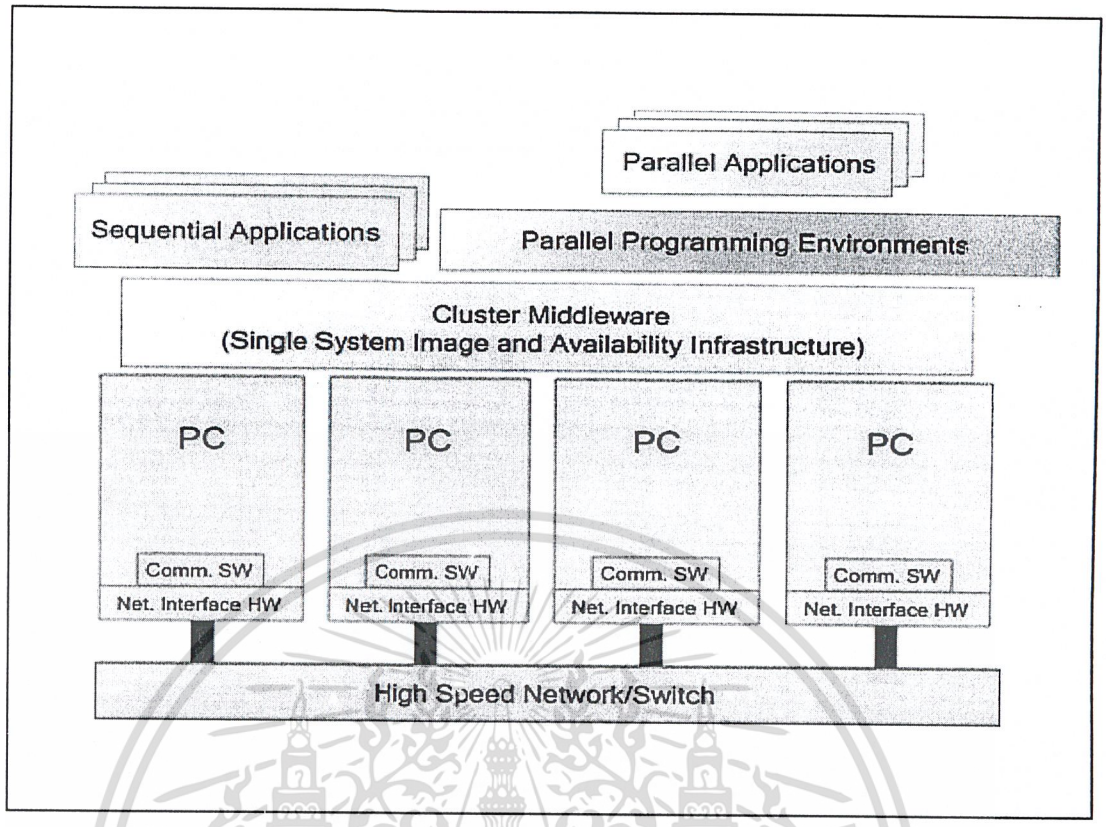
5.1 เบียวูล์ฟคลัสเตอร์ (Beowulf Cluster)

จากที่กล่าวมาแล้วในบทที่ 2 มีสถาปัตยกรรมคอมพิวเตอร์แบบขนานมากมายหลายแบบ ไม่ว่าจะเป็น PVP, MPP, SMP, DSM และคลัสเตอร์ ซึ่งวิทยานิพนธ์เล่มนี้จะเน้นศึกษาคลัสเตอร์เป็นหลัก ซึ่งคลัสเตอร์นั้นมีข้อดีมากมายอาทิเช่น ระบบสามารถสร้างได้จากอุปกรณ์ที่มีขายตามท้องตลาดไม่ต้องออกแบบฮาร์ดแวร์ขึ้นมาใหม่ เพียงเครื่องพีซีก็สามารถประกอบกันเข้าเป็นคลัสเตอร์ได้, คลัสเตอร์จะมีประสิทธิภาพต่อราคาสูง คือประสิทธิภาพค่อนข้างดีแต่ราคาต่ำทำให้เกิดความคุ้มค่าต่อการลงทุน องค์กรที่ไม่ร่ำรวยนักสามารถหาหรือสร้างคลัสเตอร์ประสิทธิภาพสูงไว้ใช้งานได้

ในปัจจุบันมีคลัสเตอร์มากมายหลายแบบหลายแพลตฟอร์ม แม้กระทั่งระบบปฏิบัติการตระกูล Windows และ Windows NT ก็สามารถสร้างเป็นคลัสเตอร์ได้ แต่ที่นิยมแพร่หลายและมีประสิทธิภาพสูงคือคลัสเตอร์บนแพลตฟอร์มยูนิกซ์ และเบียวูล์ฟคลัสเตอร์ก็จัดอยู่ในตระกูลยูนิกซ์ด้วยเช่นกัน และคลัสเตอร์แบบเบียวูล์ฟนั้นก็ยังมีแยกย่อยเป็นชนิดต่างๆกันอีก แต่จะมีส่วนหลักที่คล้ายๆกันซึ่งมีลักษณะดังนี้

5.1.1 คุณลักษณะของเบียวูล์ฟคลัสเตอร์

เบียวูล์ฟคือรูปแบบหนึ่งของการจัดสถาปัตยกรรมแบบคลัสเตอร์(รายละเอียดของสถาปัตยกรรมแบบคลัสเตอร์ได้อธิบายไว้แล้วในบทที่ 2) ซึ่งเป็นสถาปัตยกรรมคอมพิวเตอร์ที่สามารถประมวลผลแบบขนานได้ โดยคลัสเตอร์แบบเบียวูล์ฟจะประกอบไปด้วยเครื่องแม่ข่าย (Server) และเครื่องลูกข่าย (Client) ซึ่งเป็นโหนดที่สามารถประมวลผลได้ (Computation Node) และเครื่องแม่ข่ายสามารถจะเป็นโหนดประมวลผลด้วยก็ได้ โดยที่ทั้งเครื่องแม่ข่าย, เครื่องลูกข่าย หรือโหนดประมวลผลนั้นจะต่อเข้าด้วยกันผ่านระบบเครือข่ายเช่น Ethernet หรือระบบเครือข่ายมาตรฐานอื่นๆ ทั้งระบบจะประกอบด้วยผลิตภัณฑ์คอมพิวเตอร์ทั่วไปเช่น พีซี และ Ethernet NIC จะไม่มีการออกแบบหรือสร้างฮาร์ดแวร์ใดๆขึ้นมาใหม่ทั้งสิ้น ทุกๆโหนดรันด้วยระบบปฏิบัติการใช้ลินุกซ์ มีการใช้ไลบรารี่เมสเซจพาสซิ่งเช่น PVM หรือ MPI สำหรับจัดการการประมวลผลแบบขนาน โดยมีตัวจัดการสภาพแวดล้อมของการประมวลผลงานแบบขนานเป็นตัวคั่นกลางระหว่างซอฟต์แวร์และระบบปฏิบัติการ และทำให้มองภาพรวมของทั้งระบบเป็นหนึ่งเดียวกัน (Single System Image) และซอฟต์แวร์นั้นจะไม่สามารถเห็นและเข้าถึงรายละเอียดของฮาร์ดแวร์ระดับล่างได้ แต่จะมองเห็นเสมือนเป็นเครื่องเครื่องเดียว และมีการขอใช้ทรัพยากรของระบบโดยติดต่อกับตัวจัดการสภาพแวดล้อมแบบขนานและระบบปฏิบัติการเท่านั้น ดังแสดงในรูปที่ 5.1



รูปที่ 5-1 แบบจำลองคลัสเตอร์แบบเบียวูล์ฟ

5.1.2 เซิร์ฟเวอร์โหนด, ไคลเอนต์โหนด และ โหนดคำนวณ

ในระบบเบียวูล์ฟคลัสเตอร์ซึ่งประกอบไปด้วยเครื่องพีซีจำนวนหลายๆเครื่องนั้น แต่ละเครื่องเราจะเรียกว่าโหนด และโหนดยังแบ่งออกเป็นเซิร์ฟเวอร์โหนด (Server Node, ไคลเอนต์โหนด (Client Node) และโหนดคำนวณ (Computation Node)

- **เซิร์ฟเวอร์โหนด** คือโหนดที่ทำหน้าที่ควบคุมและจัดการระบบให้ทำงานได้อย่างถูกต้อง การสั่งงานทุกอย่างจะผ่านเซิร์ฟเวอร์โหนดเสมือนเป็นศูนย์กลางสั่งงาน และคอยติดรับคำสั่งจากผู้ใช้งานด้วย เซิร์ฟเวอร์โหนดสามารถมีได้มากกว่า 1 ตัว เช่นในบางระบบมีเซิร์ฟเวอร์โหนด ที่ทำหน้าที่กระจายงานออกไปสู่ไคลเอนต์โหนด และเซิร์ฟเวอร์โหนดที่ทำหน้าที่ติดต่อรับคำสั่งจากผู้ใช้งานและเป็นเกตเวย์ออกสู่ภายนอกระบบ แยกจากกันอย่างชัดเจน เพื่อให้เซิร์ฟเวอร์โหนดแต่ละตัวทำงานของมันได้อย่างเต็มที่และมีประสิทธิภาพ ไคลเอนต์โหนด
- **ไคลเอนต์โหนด** คือโหนดที่ถูกควบคุมจากเซิร์ฟเวอร์โหนดให้ทำงานตามที่กำหนด ไคลเอนต์โหนดนี้จะเป็นเสมือนแขนขาของระบบ และมีจำนวนมากที่สุดด้วย
- **โหนดคำนวณ** คือโหนดที่มีหน้าที่ในการคำนวณและประมวลผลงานที่ถูกจ่ายออกมาจากเซิร์ฟเวอร์โหนด โดยทั่วไปแล้วไคลเอนต์โหนดจะเป็นโหนดคำนวณ ส่วนเซิร์ฟเวอร์โหนดนั้นอาจจะเป็นหรือไม่เป็นก็ได้ ซึ่งในระบบที่ต้องการประสิทธิภาพสูง จะทำการแยก

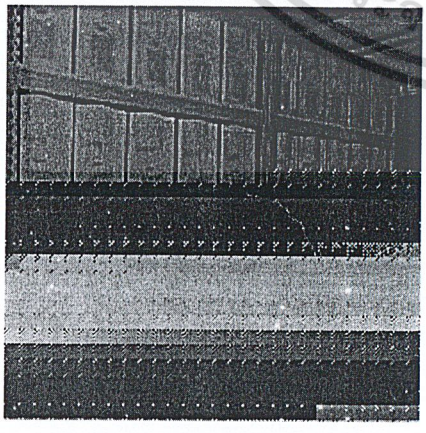
เซิร์ฟเวอร์เพื่อทำงานควบคุมอย่างเดียวไม่ได้มีการคำนวณใดๆทั้งสิ้น แต่ก็มีบางระบบเช่นกันที่เซิร์ฟเวอร์โหนดเป็นโหนดคำนวณด้วยในตัว

5.1.3 ข้อดีข้อเสียของเบียวูล์ฟคลัสเตอร์

จุดประสงค์หลักของการออกแบบเบียวูล์ฟคลัสเตอร์นั้นคือเน้นสร้างระบบคอมพิวเตอร์ที่มีประสิทธิภาพสูงแต่ราคาประหยัด ดังนั้นจึงเลือกใช้อุปกรณ์มาตรฐานต่างๆ ที่มีปริมาณการซื้อขายในท้องตลาดมากทำให้ราคาต่ำ เบียวูล์ฟคลัสเตอร์บางแบบนั้นทำเป็นแบบดิสค์เลสด้วย คือไม่ต้องใช้หน่วยความจำสำรอง ทุกโหนดคำนวณ จะนำระบบไฟล์ทั้งหมดเก็บไว้ที่เครื่องแม่ข่ายแล้วทำการแชร์ไฟล์เหล่านั้นผ่านเครือข่ายให้เครื่องลูกข่ายใช้ การบู๊ตก็จะกระทำผ่านเครือข่ายทั้งหมด ข้อดีคือมีการจัดเก็บระบบไฟล์ไว้ที่เดียว ทำให้ง่ายต่อการจัดการ อีกทั้งยังประหยัดค่าอุปกรณ์ความจำสำรองต่างๆเช่นฮาร์ดดิสก์ ซีดีรอมเป็นต้น ในบางระบบนั้นโหนดคำนวณจะมีเพียงแต่เมนบอร์ด ซีพียู หน่วยความจำ และแหล่งจ่ายไฟเท่านั้น ซึ่งเราจะเรียกโหนดเหล่านี้ว่าเฮดเลส ซึ่งการตัดอุปกรณ์รอบข้างเช่นจอ, คีย์บอร์ด, เมาส์ ทิ้งไป โหนดนั้นก็ยังสามารถคำนวณต่อไปได้ อีกทั้งยังทำให้ลดค่าใช้จ่ายไปได้มากด้วย ส่วนข้อเสียของมันคือการผลักภาระทั้งหมดไปให้กับเครือข่าย และการแชร์ไฟล์ผ่านเครือข่ายซึ่งมีโอเวอร์เฮดสูง ทำให้เครือข่ายต้องมีความเร็วที่มากพอจึงจะเกิดประสิทธิภาพสูงสุด, เบียวูล์ฟคลัสเตอร์บางระบบที่เน้นประสิทธิภาพ จะมีการใช้เครือข่ายความเร็วสูงเช่น Gigabit Ethernet หรือ Fiber Optic เลยทีเดียว ข้อดีอีกอย่างหนึ่งของเบียวูล์ฟคลัสเตอร์ก็คือ มีการใช้ไลบรารี่มาตรฐานต่างๆ ซึ่งแจกจ่ายกันฟรี รวมถึงใช้ระบบปฏิบัติการลินุกซ์ ซึ่งเป็นระบบที่มีประสิทธิภาพสูงและฟรีอีกเช่นกัน และเมื่อพิจารณาถึงราคาอุปกรณ์ในระบบแล้ว เบียวูล์ฟคลัสเตอร์ถือได้ว่า เป็นระบบประสิทธิภาพสูงในราคาที่ต่ำมาก การขยายระบบออกไปก็ทำได้ง่าย เบียวูล์ฟคลัสเตอร์บางระบบมีจำนวนโหนดถึง 70 –120 โหนดเลยทีเดียว

5.2 การออกแบบ

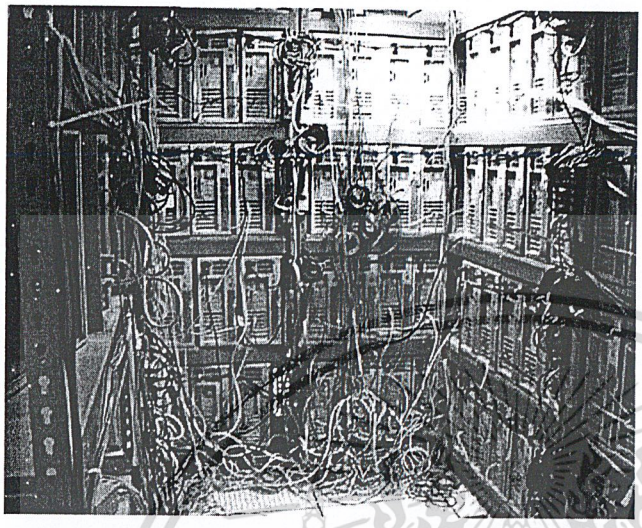
5.2.1 ตัวอย่างของระบบที่ใช้งานจริง



รูปที่ 5-2 Avalon Cluster

อาวาลอนซุปเปอร์คอมพิวเตอร์ เป็นสถาปัตยกรรมคลัสเตอร์แบบเบียวูล์ฟที่มีคุณสมบัติดังนี้

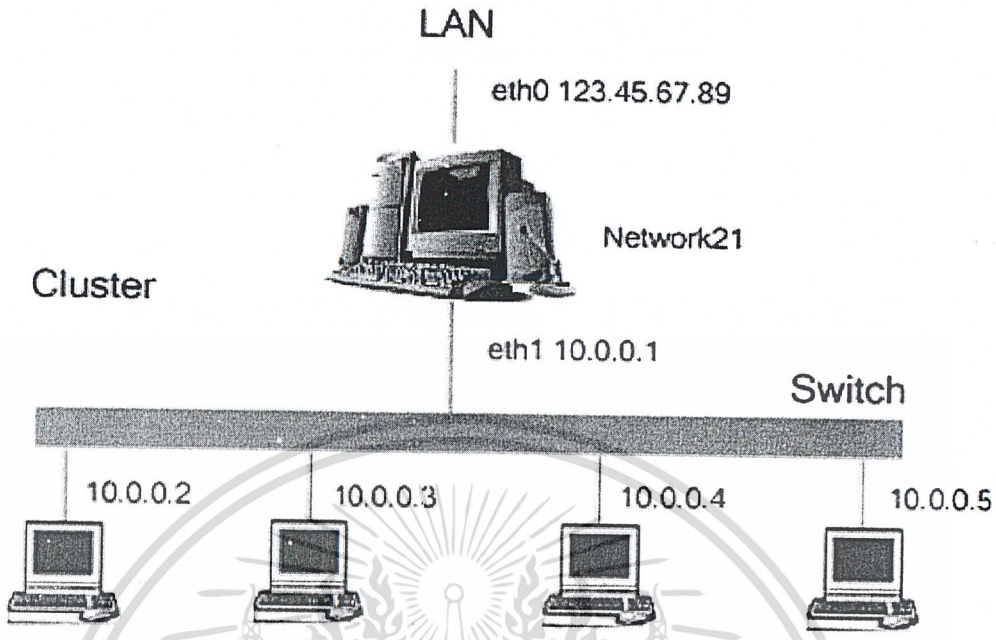
- ใช้โปรเซสเซอร์แบบ DEC Alpha จำนวน 70 ตัว
- ประสิทธิภาพในการคำนวณเท่ากับ 10 Gflops



รูปที่ 5-3 ด้านหลังของคลัสเตอร์ เป็นการเชื่อต่อของเครือข่ายไฟเบอร์ออพติก

รูปที่ 5-4 เทอมินอล,เซิร์ฟเวอร์, และไคลเอนต์ ในสภาพใช้งานจริง

5.2.2 การสร้างคลัสเตอร์อย่างง่าย



รูปที่ 5-5 การออกแบบสร้างคลัสเตอร์โดยใช้ลินุกซ์พีซี ผ่านเครือข่ายอีเทอร์เน็ต

จากรูปที่ 5.5 เป็นการออกแบบสร้างคลัสเตอร์ที่ประกอบจากเซิร์ฟเวอร์ 1 เครื่องและไคลเอนต์จำนวน 4 เครื่อง ผ่านเครือข่ายอีเทอร์เน็ต โดยเชื่อมต่อเข้าสวิทช์เพื่อประสิทธิภาพของการรับส่งข้อมูลในเครือข่าย โดยเครื่องไคลเอนต์จะถูกกำหนดให้เป็นแบบไพรเวทไอพีแอดเดรส (private ip address) เป็นเครือข่ายภายในที่ไม่สามารถเข้าถึงได้จากเครือข่ายภายนอก เครื่องเซิร์ฟเวอร์นอกจากจะมีการเชื่อมต่อกับเครือข่ายภายในแล้ว ตัวเซิร์ฟเวอร์เองยังทำหน้าที่เป็นเกตเวย์ (Gateway) ออกไปสู่เครือข่ายภายนอกอีกด้วย โดยเซิร์ฟเวอร์จะมี NIC 2 ตัว ตัวหนึ่งกำหนดไอพีจริงให้เพื่อทำการติดต่อกับเครือข่ายภายนอก ส่วนอีกตัวหนึ่งกำหนดให้เป็นไพรเวทไอพีแอดเดรสเพื่อติดต่อกับเครือข่ายภายในคลัสเตอร์เอง การเพิ่มไคลเอนต์โหนดสามารถทำได้ง่ายดายโดยการเพิ่มเครื่องไคลเอนต์และเชื่อมต่อเข้ากับเครือข่ายตรงจุดต่อสวิทช์และกำหนดไอพีแอดเดรสไม่ให้ซ้ำกับโหนดใดๆที่อยู่ในคลัสเตอร์อยู่แล้ว

5.2.3 ออกแบบคลัสเตอร์ต้นแบบ

เราจะเริ่มออกแบบคลัสเตอร์ขึ้นเพื่อใช้งานจริงโดยดูจากอุปกรณ์และทรัพยากรณ์ที่มีอยู่นี้

1st PC

- CPU Cellron 566 Mhz
- Memory 96 Mb SD-RAM
- Harddisk 10.2 Gb
- Floppy Drive, CD-ROM Drive

- VGA card, Monutor, Mouse Keyboard

2nd PC

- CPU Pentium MMX 166 MHz
- Memory 64 MB SD-RAM
- Floppy Drive, CD-ROM Drive
- VGA card, Monitor, mouse, keyboard

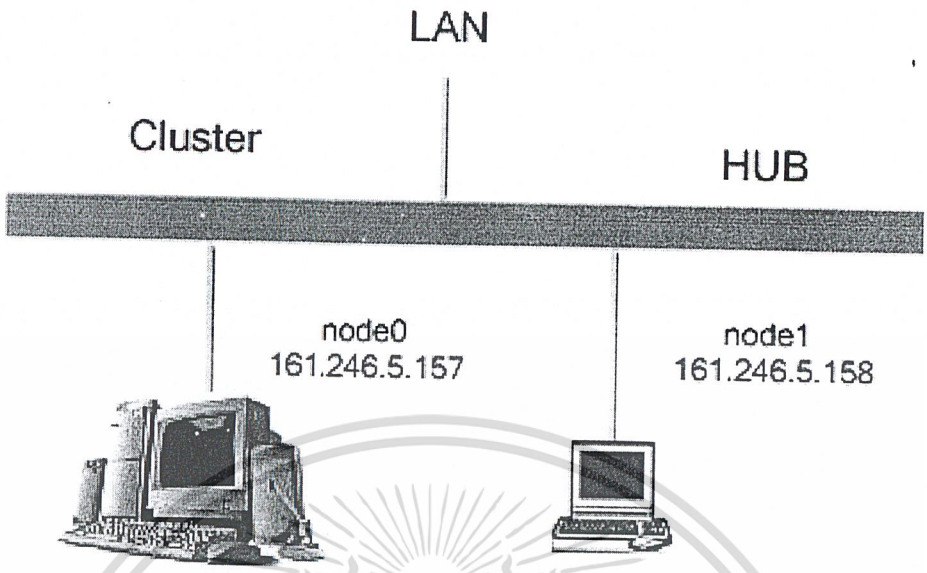
NIC

- 2 x 10baseT Ethernet Card

Network

- UTP cable
- HUB 10baseT
- Connect to Internet

เนื่องจากว่าเราไม่สามารถกำหนดไพรเวทไอพีแอดเดรสให้โหนดได้ เพราะเครือข่ายที่ใช้งานมีผู้ใช้คนอื่นร่วมอยู่ด้วย (Network Address 161.246.5.0) ดังนั้นจึงกำหนดไอพีแอดเดรสจริงให้กับทุกโหนด และเราจะให้เครื่องพีซีเครื่องแรกเป็นเซิร์ฟเวอร์โหนด ส่วนพีซีเครื่องที่สองเป็นไคลเอนต์โหนด และเนื่องจากว่ามีเครื่องพีซีน้อย จึงให้เซิร์ฟเวอร์เป็นโหนดคำนวณด้วย เราจึงกำหนดโครงสร้างของฮาร์ดแวร์อย่างคร่าวๆได้ดังรูปที่ 5.6



รูปที่ 5-6 โครงสร้างของคลัสเตอร์ต้นแบบ

5.3 ขั้นตอนการสร้าง

5.3.1 ติดตั้งระบบปฏิบัติการ

ติดตั้งระบบปฏิบัติการ Linux Redhat version 6.2 บน Server node และต้องมีพื้นที่ว่างของ harddisk เหลือพอสำหรับ NFS-root file system ของแต่ละ client node (ประมาณ 500 KB ต่อ client 1 node) ซึ่งในระบบเริ่มต้นจะติดตั้ง client node เพียง node เดียว และ Kernel ต้องสนับสนุน RARP โพรโตคอลด้วย

5.3.2 เตรียมการบู๊ตผ่านเครือข่าย

การบู๊ตผ่านเครือข่ายสามารถเลือกได้สองวิธีคือ การบู๊ตผ่านรอม (ROM) และบู๊ตผ่านฟลอปปี้ดิสก์

5.3.2.1 บู๊ตผ่านรอม (ROM)

เป็นการบู๊ตเครื่องคอมพิวเตอร์จากรอมที่อยู่ในการ์ดเน็ตเวิร์ก โดยการ์ดเน็ตเวิร์กนี้บางรุ่นจะมีรอมฝังอยู่ ซึ่งภายในรอมจะบรรจุโปรแกรมที่สนับสนุนโปรโตคอลแบบ bootp เมื่อคอมพิวเตอร์บู๊ตขึ้นมาและถ้าตรวจพบว่ามีรอมอยู่ภายในการ์ดเน็ตเวิร์กแล้ว จะทำการกระโดดการทำงานไปบู๊ตจากรอมที่อยู่ในการ์ดเน็ตเวิร์กแทน เมื่อรอมเริ่มทำงานจะทำการติดต่อผ่านเซิร์ฟเวอร์ (ที่ทำ

การติดตั้งไว้แล้ว) โดยใช้โปโตรคอลแบบ bootp และเข้าสู่กระบวนการของการบู๊ตต่อไป แต่จะไม่ขอกล่าวรายละเอียดในหัวข้อนี้ เพราะการบู๊ตจากรอมเป็นวิธีการที่ล้าสมัยแล้วและเสียค่าใช้จ่ายมากกว่าด้วย (เพราะการ์ดเน็ตเวิร์กจะต้องมีซอกเก็ตสำหรับเสียบ EPROM ซึ่งปัจจุบันไม่ค่อยมีผู้ผลิตทำการ์ดเน็ตเวิร์กรุ่นที่มีซอกเก็ตไว้เสียบ EPROM แล้ว หรือมีก็แพงกว่า) อีกทั้งการบรรจุโปรแกรมลงรอมนั้นมีขั้นตอนที่ยุ่งยากและละเอียดพอสมควร ต้องใช้บุคคลที่มีความรู้ในการโปรแกรม สำหรับใช้ใส่ลงไปในรอมด้วย

5.3.2.2 บู๊ตผ่านฟลอปปี้ดิสก์

เป็นการบู๊ตจากเคอร์เนล(kernel) ที่อยู่ในฟลอปปี้ดิสก์ ที่สนับสนุนโปรโตคอลแบบ rarp (reverse address resolution protocol) โดยเมื่อเริ่มบู๊ตขึ้นมาเคอร์เนลจากแผ่นจะถูกโหลดเข้าไปไว้ในหน่วยความจำและทำหน้าที่เป็นระบบปฏิบัติการชั่วคราว หลังจากนั้นเคอร์เนลจะเริ่มใช้โปรโตคอล rarp โดยมีขั้นตอนดังนี้

- บรอดคาสท์ (broadcast) ข้อความกระจายออกไปในเครือข่ายเพื่อร้องบริการจากเซิร์ฟเวอร์
- เซิร์ฟเวอร์ที่เปิดบริการอยู่ในขณะนั้นตอบกลับมา
- เคอร์เนลทำการส่งแม็คแอดเดรส (mac address) กระจายออกไปในเครือข่าย ซึ่งแม็คแอดเดรสนี้เป็นตัวเลขขนาด 6 ไบต์ ซึ่งจะติดมากับ NIC (Network Interface Card) ทุกตัวอยู่แล้ว และตัวเลขนี้จะเป็นหนึ่งเดียวคือ ไม่ซ้ำกับตัวเลขบน NIC ตัวอื่นเลย และเราอาจเรียกตัวเลข 6 ไบต์นี้ว่าแม็คแอดเดรสหรือฮาร์ดแวร์แอดเดรสก็ได้
- เซิร์ฟเวอร์ได้รับแม็คแอดเดรสแล้วทำการเปรียบเทียบแม็คแอดเดรสที่ได้มา กับ rarp table ที่อยู่บนเซิร์ฟเวอร์แล้วทำการส่งไอพีแอดเดรสตอบกลับไป
- เคอร์เนลได้รับไอพีแอดเดรสจากเซิร์ฟเวอร์แล้วทำการคอนฟิก NIC ให้ใช้ไอพีแอดเดรสที่ได้รับมาและจะใช้โปรโตคอลทีซีพีไอพี ในการติดต่อกับเซิร์ฟเวอร์
- หลังจากนั้นเคอร์เนลจะแมพตัวเองไปเป็นไคลเอนต์ของเซิร์ฟเวอร์เพื่อเข้าไปเมาท์ (mount) ระบบไฟล์จากเซิร์ฟเวอร์แล้วจึงเข้าสู่กระบวนการบู๊ตลินุกซ์ปกติ เพียงแต่ว่าระบบไฟล์ของไคลเอนต์นั้นไปอยู่บนเซิร์ฟเวอร์แค่นั้นเอง

5.3.3 จัดเตรียมพื้นที่ใช้งานสำหรับการบู๊ตจากเครือข่าย

ในการบู๊ตจากเครือข่ายเราต้องต้องจัดเตรียมพื้นที่ใช้งานสำหรับเป็นระบบไฟล์ของไคลเอนต์บนฝั่งเซิร์ฟเวอร์ก่อนด้วยการเรียกสคริปต์ sdct (set disk-less client template) ขึ้นทำงานที่เครื่องเซิร์ฟเวอร์เพื่อสร้าง template directory (ซึ่งจะอยู่ที่ /tftpboot/Template) โดยสามารถหาสคริปต์นี้ได้จาก ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils/disk-less/sdct (หรือที่ภาคผนวกของวิทยานิพนธ์เล่มนี้)

template directory นี้จะถูกสร้างขึ้นจากสคริปต์และอยู่ที่ /tftpboot/Template ซึ่งจะเป็นแม่แบบสำหรับระบบไฟล์ต่างๆของไคลเอนต์ เช่น root file system ของไคลเอนต์ และไฟล์ต่างๆเท่าที่จำเป็นสำหรับการบู๊ตผ่านเครือข่าย บางส่วนของไฟล์นั้นสำเนามาจากระบบไฟล์บนเครื่องเซิร์ฟเวอร์และบาง

ส่วนนั้นเป็นการลิงค์ (link) มาจากไฟล์บนเครื่องเซิร์ฟเวอร์เท่านั้น การลิงค์ทำขึ้นเพื่อชี้ไปยังไฟล์ต้นฉบับจริงๆซึ่งอยู่ที่เซิร์ฟเวอร์ เมื่อไฟล์ที่อยู่บนเซิร์ฟเวอร์นั้นเปลี่ยนแปลง ไฟล์ที่อยู่บนไคลเอนต์ที่ลิงค์ไปยังไฟล์นั้นก็จะเปลี่ยนตามด้วย ทำให้การเปลี่ยนแปลงไฟล์ที่เซิร์ฟเวอร์เพียงครั้งเดียว มีผลกับไฟล์บนไคลเอนต์ทุกตัวด้วย (มีประโยชน์กรณีที่ไฟล์นั้นเป็นไฟล์ที่ต้องใช้ข้อมูลร่วมกันหรือเหมือนกันทั้งเซิร์ฟเวอร์และไคลเอนต์ทุกตัว เช่น /etc/passwd บนเซิร์ฟเวอร์และไคลเอนต์ควรจะเป็นชุดเดียวกัน) ซึ่งจะทำให้การจัดการง่ายดายขึ้น

เมื่อทำการรันสคริปต์ sdct บนฝั่งเซิร์ฟเวอร์แล้ว สคริปต์จำทำการสร้าง template directory ซึ่งอยู่ที่ /tftpboot/Template ให้เราลองเข้าไปดูไฟล์ในไดเรกทอรีนั้นโดยใช้คำสั่ง

```
ls –l /tftpboot/Template
```

เราจะพบว่ามีทั้งไฟล์ปกติที่สำเนามาจากไฟล์ของเซิร์ฟเวอร์ (/tftpboot/template นี้ก็อยู่บนเซิร์ฟเวอร์เช่นกัน แต่จะใช้เป็นระบบไฟล์บนไคลเอนต์) และไฟล์ที่ลิงค์มาจากเซิร์ฟเวอร์ ซึ่งไฟล์ทั้งสองแบบนี้จะมีความแตกต่างกันคือ ไฟล์ที่สำเนามาจากเซิร์ฟเวอร์จะใช้แยกเป็นอิสระต่อกันบนแต่ละโหนด เช่นไฟล์คอนฟิกฮาร์ดแวร์ต่างๆซึ่งแต่ละโหนดจะไม่เหมือนกัน และไฟล์ที่ลิงค์มาจากเซิร์ฟเวอร์ซึ่งจำเป็นต้องใช้ร่วมกันในทุกๆโหนดเช่น /etc/passwd ซึ่งตารางข้างล่างจะแสดงไฟล์ที่เป็นอิสระต่อกันและไฟล์ที่ใช้ร่วมกันในแต่ละโหนด ในไดเรกทอรี /tftpboot/Template

ไฟล์ที่ทำสำเนามาจากเซิร์ฟเวอร์	ไฟล์ที่ลิงค์มาจากเซิร์ฟเวอร์ และใช้ร่วมกันทุกโหนด	ไฟล์ที่สร้างขึ้นใหม่
/bin	/proc	/home
/boot	/usr	/tmp
/dev	/etc/passwd	
/etc	/etc/group	
/lib	/etc/issue	
/mnt	/etc/issue.net	
/root	/etc/profile	
/sbin	/etc/bashrc	
/var	/etc/hosts	
	/etc/hosts.equiv	

ตารางที่ 5-1 ระบบไฟล์ใน /tftpboot/Template

5.3.4 สร้างแผ่นบู๊ตสำหรับไคลเอนต์ (client boot floppy)

สร้างแผ่นบู๊ตเพื่อใช้บู๊ตผ่านเน็ตเวิร์กโดยการคอมไพล์เคอร์เนล เป็นแบบโมโนลิธิก (monolithic) ซึ่งเป็นการฝังโมดุล (module) เข้าไปในเคอร์เนล และจะต้องแน่ใจว่า kernel สนับสนุน NFS, NFS-root file system , bootp และ rarp โพรโตคอล โดยตอบ 'y' สำหรับหัวข้อ CONFIG_ROOT_NFS, CONFIG_RNFS_BOOTP, CONFIG_RNFS_RARP ในตอน Compile kernel ด้วย

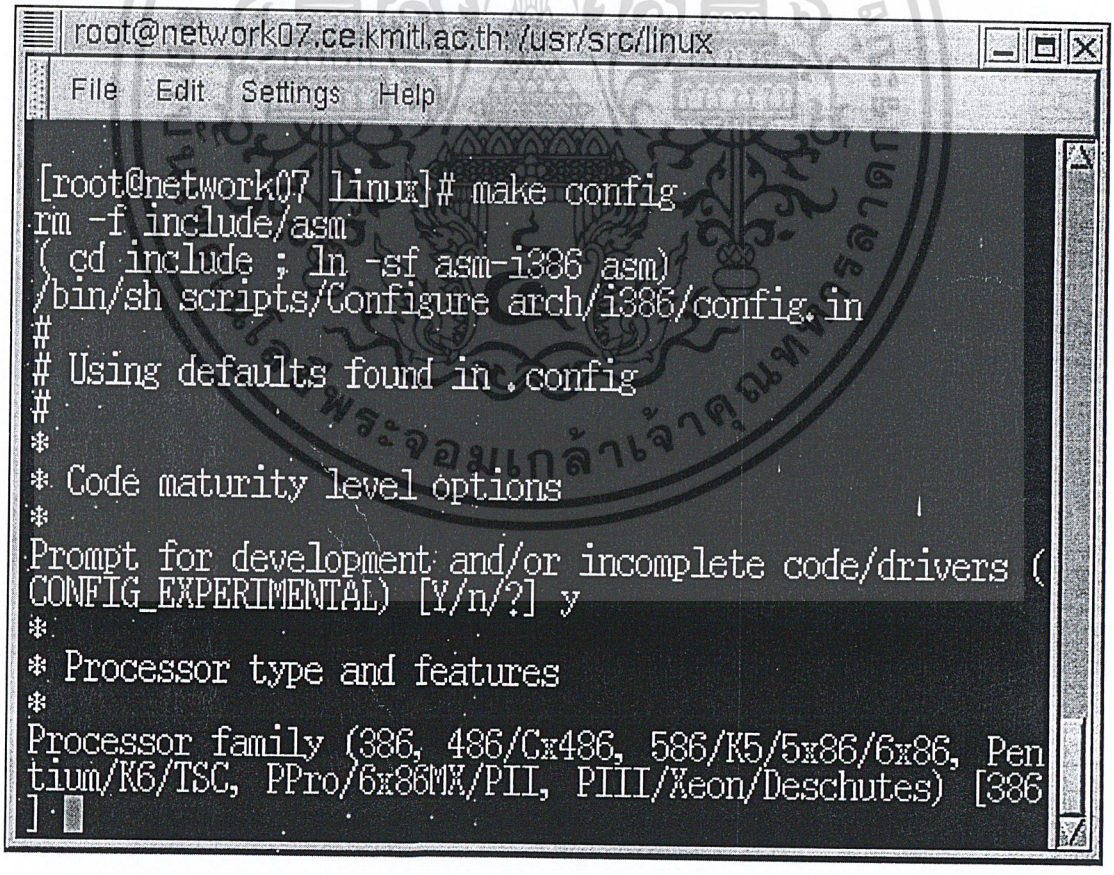
5.3.4.1 การปรับแต่งเคอร์เนลสำหรับแผ่นบู๊ต

การปรับแต่งเคอร์เนลบนลินุกซ์นั้นทำได้ 4 ทางคือ

1 **กำหนดโดยใช้โปรแกรมถามตอบทีละข้อ** โดยเข้าไปที่ไดเรกทอรี /usr/src/linux แล้วใช้คำสั่ง

```
# make config
```

ระบบจะถามคำถามขึ้นมาว่าจะให้เคอร์เนลมีคุณสมบัติอะไรบ้าง โดยต้องเลือกตอบว่าจะให้มี (Y,Yes), หรือไม่มี (N,No) หรือทำเป็นโมดูล (M,Module) หากมีข้อสงสัยคุณสมบัติโปรแกรมที่ถามขึ้นมานั้นทีรายละเอียดอย่างไร ให้กดคีย์ ? จากนั้นโปรแกรมจะแสดงรายละเอียดขึ้นมา



รูปที่ 5-7 การปรับแต่งเคอร์เนล โดยใช้โปรแกรมถามตอบทีละข้อ

2 กำหนดโดยใช้โปรแกรมแบบเท็กซ์เมนู ให้เข้าไปที่ไดเร็กทอรี /usr/src/linux แล้วใช้คำสั่ง

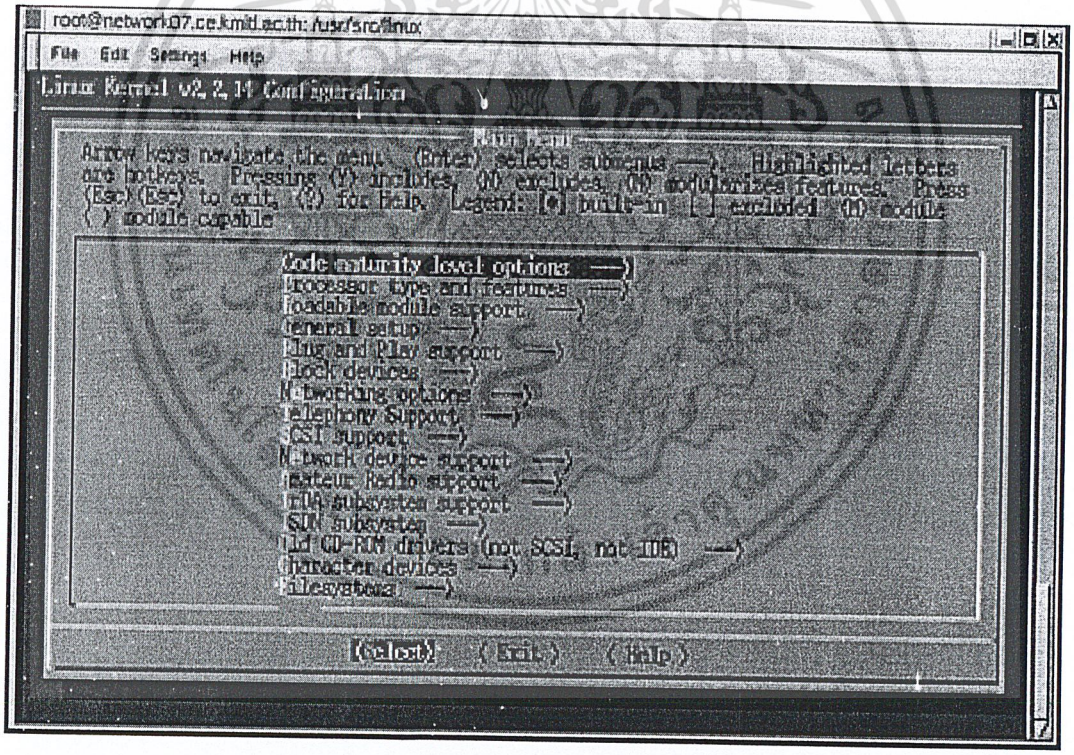
```
# make menuconfig
```

วิธีนี้ดูจะเข้าใจง่ายกว่าวิธีแรก เพราะจะมีลักษณะเป็นวินโดว์ที่มีตัวเลือกและเปลี่ยนการแก้ไขสลับไปมาได้ โดยในวินโดว์แรกจะเป็นเมนูหลัก ที่จะแสดงกลุ่มของคุณลักษณะต่างๆของเคอร์เนลซึ่งเราสามารถเลื่อนแถบควบคุมขึ้นลง ไปตามกลุ่มต่างๆเหล่านั้นได้ ส่วนแถบควบคุมด้านล่างจะบอกว่า เราต้องการจะทำอะไรกับตัวเลือกในขณะนั้น ซึ่งจะมีตัวเลือกคือ

<Select> ยืนยันการเลือก

<Exit> ยกเลิกหรือออกจากหน้าต่างปัจจุบัน

<Help> ขอรายละเอียดของตัวเลือกปัจจุบัน



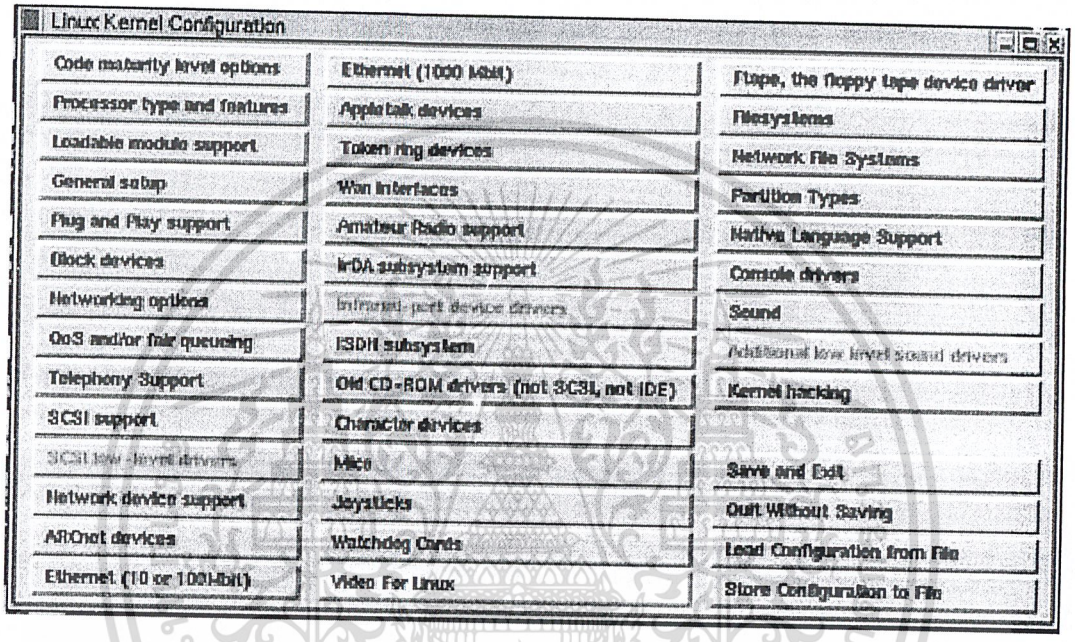
รูปที่ 5-8 การปรับแต่งเคอร์เนลโดยใช้โปรแกรมแบบเทกซ์เมนู

เมื่อทำการเลือกคุณลักษณะของเคอร์เนลเสร็จเรียบร้อย โปรแกรมจะถามว่าให้บันทึกข้อมูลของเคอร์เนลใหม่หรือไม่ให้ตอบ “Yes”

3 การกำหนดโดยใช้รูปแบบกราฟิก วิธีนี้ต้องใช้งานภายใต้ X Window โดยพิมพ์คำสั่ง

make xconfig

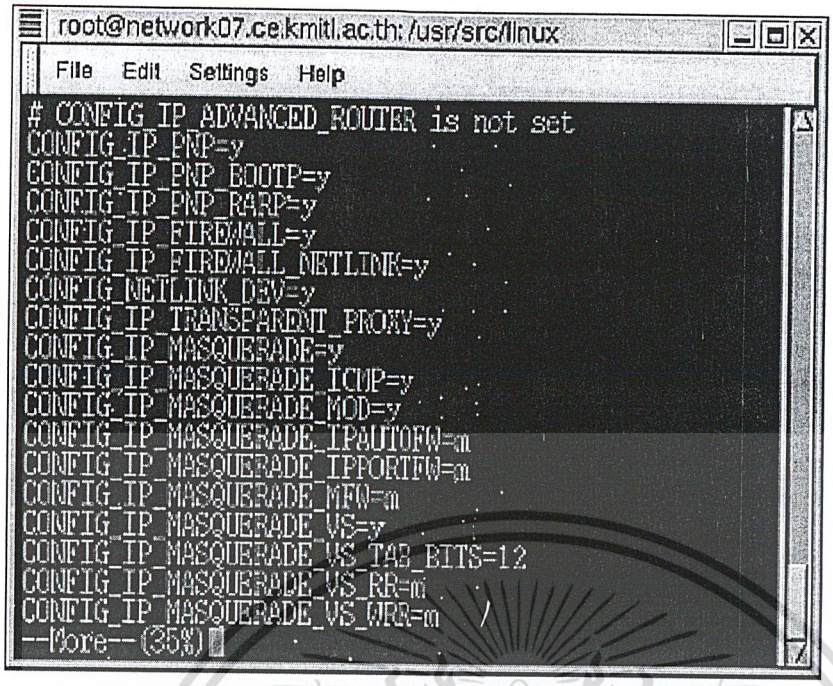
การกำหนดในลักษณะนี้จะทีรูปแบบการใช้งานคล้ายกับวิธีที่ 2 แต่จะใช้งานได้ง่ายกว่าคือ เราสามารถใช้เมาส์เลือกชี้และคลิกเมนูต่างๆที่ต้องการได้ เมื่อคลิกเมนูแล้ว ก็จะปรากฏเป็นรายการให้เลือกคอนฟิกภายใต้เมนูนั้นๆ ซึ่งมีให้เลือกได้ 3 แบบคือ รวมลงไปในเคอร์เนล (คลิกที่ปุ่ม y), ทำเป็นโมดุล (คลิกที่ปุ่ม m) หรือจะไม่รวมเข้ากับเคอร์เนล (คลิกที่ปุ่ม n) แล้วคลิกที่ปุ่ม "Save and Exit"



รูปที่ 5-9 การปรับแต่งเคอร์เนลโดยใช้รูปแบบกราฟิก

4 การกำหนดเคอร์เนลโดยตรง

เราสามารถทำได้โดยการใช้โปรแกรมเอดิเตอร์เช่น vi หรือ pico โดยทำการแก้ไขข้อมูลในไฟล์ /usr/src/linux/.config โดยตรง



รูปที่ 5-10 การปรับแต่งเคอร์เนลโดยตรง

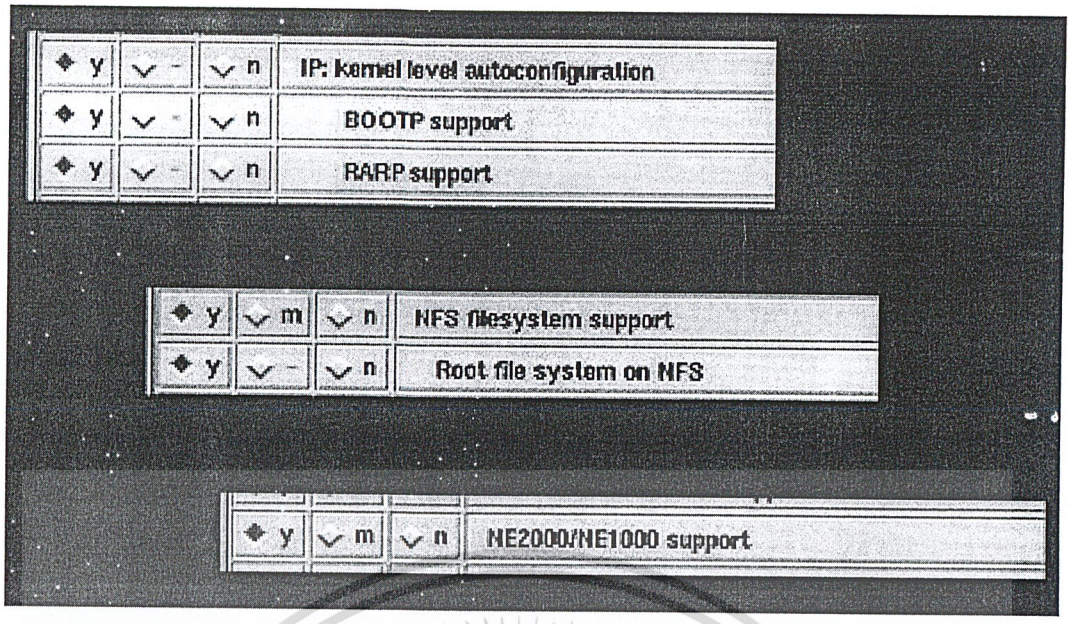
วิธีกำหนดแบบนี้ ผู้ใช้จะต้องมีความรู้เกี่ยวกับความหมายของตัวเลือกที่แสดงออกมาในแต่ละชื่อด้วย วิธีกำหนดคุณลักษณะของเคอร์เนลจะเป็นดังนี้

CONFIG_ANY=y กำหนดให้รวมเข้ากับเคอร์เนล

CONFIG_ANY=n กำหนดไม่ให้รวมเข้ากับเคอร์เนล

CONFIG_ANY=m กำหนดให้ทำเป็นโมดุล

เพื่อความสดวกเราจะใช้การปรับแต่งเคอร์เนลแบบที่ 3 โดยเลือกโมดุลเฉพาะที่จำเป็นฝังลงไปในเคอร์เนล ส่วนโมดุลที่ไม่ได้ใช้ให้เอาออก เพื่อลดขนาดของเคอร์เนลให้เล็กที่สุดจะได้ประหยัดเนื้อที่และการบู๊ตจะได้เป็นไปอย่างรวดเร็ว โดยเคอร์เนลจะต้องสนับสนุน NFS, NFS-root file system , bootp, rarp โพรโตคอล และต้องมีโมดุลของการ์ดเน็ตเวิร์กของเครื่องไคลเอนต์ด้วย



รูปที่ 5-11 การปรับแต่งเคอร์เนลให้สนับสนุน NFS, NFS-root file system , bootp, rarp และการ์ดเน็ตเวิร์ก

5.3.4.2 การคอมไพล์เคอร์เนล

ทำการคอมไพล์เคอร์เนลที่เราได้ทำการปรับแต่งไปแล้วโดยใช้คำสั่งดังนี้

```
# cd /usr/src/linux
# make dep && make clean && make zImage
```

โดยเมื่อคำสั่งทำงานสิ้นสุดแล้วจะได้เคอร์เนลที่ถูกคอมไพล์แล้วชื่อ zImage โดยอยู่ในไดเรกทอรี /usr/src/linux/arch/i386/boot แต่ถ้าเคอร์เนลที่ได้มีขนาดใหญ่เกินไป ต้องเปลี่ยนจากคำว่า zImage ไปเป็น bzImage แทน

5.3.4.3 การเปลี่ยน root device และการนำเคอร์เนลลงแผ่นฟลอปปี้ดิสก์

ทำการเปลี่ยน root device ให้เป็น NFS-root จากนั้นคัดลอกลง floppy disk ตามคำสั่งต่อไปนี้

```
mknod /div/nfsroot b 0 255
cd /usr/linux/arch/i386/boot
rdev zImage /dev/nfsroot
dd if=zImage of=/dev/fd0
```

นำแผ่นดิสก์ใส่ที่ฟลอปปี้ไดรฟ์แล้วพิมพ์ตามคำสั่งข้างบน ก็จะได้แผ่นบู๊ตสำหรับทำการบู๊ตผ่านเครือข่าย

5.3.5 สร้างระบบไฟล์สำหรับไคลเอนต์

ทำการเรียกสคริปต์ adcn (add diskless client node) ขึ้นมาทำงานที่ฝั่งเซิร์ฟเวอร์ โดยสามารถหาสคริปต์นี้ได้จาก ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils/disk-less/adcn ซึ่งสามารถเรียกใช้งานสคริปต์ได้ดังนี้

```
# chmod 755 adcn
# ./adcn option
```

สคริปต์นี้จะทำการเพิ่มไคลเอนต์โหนดเข้าไปในคลัสเตอร์ โดยจะทำการสร้างระบบไฟล์สำหรับรองรับการบู๊ตผ่านเครือข่าย และปรับคอนฟิกต่างๆของฮาร์ดแวร์และเครือข่าย สคริปต์นี้มีประโยชน์อย่างมาก ช่วยลดเวลาในการทำสำเนาและปรับแต่งไฟล์ต่างๆ โดยคำสั่งต่างๆในสคริปต์นี้สามารถดูได้ที่ภาคผนวก ส่วนออพชั่นของสคริปต์นี้มีรายละเอียดดังแสดงในส่วนต่อไป

Option	คำอธิบาย
-f	หากสคริปต์นี้ฟ้องว่าไม่สนับสนุนลินุกซ์เวอร์ชั่นอื่นนอกจากเวอร์ชั่น 5.2 (ซึ่งเป็นเวอร์ชั่นใหม่สุดในขณะนั้น) ให้ใช้ออพชั่น – f เพื่อทำการรันสคริปต์นี้โดยไม่สนใจเวอร์ชันของลินุกซ์
-i ip_address	ไอพีแอดเดรสของไคลเอนต์โหนดที่เราจะทำการเพิ่มเข้าไป
-m mac_address	แมคแอดเดรส (mac address) ของไคลเอนต์โหนด
-l	เป็นการฟัง RARP request จากอินเตอร์เฟสการ์ดที่ระบุในออพชั่น –D เพื่อใช้สำหรับดักจับแมคแอดเดรส แต่ถ้าแมคแอดเดรสได้ระบุไว้ในออพชั่น – m แล้ว ก็จะไม่สนใจค่าในออพชั่น – m นั้น
-c client_name	ชื่อเครื่องของไคลเอนต์โหนดที่เราจะทำการเพิ่มเข้าไปในคลัสเตอร์
-d domain_name	ชื่อโดเมนของไคลเอนต์ ถ้าไม่ได้ระบุออพชั่นนี้ จะถือว่าไคลเอนต์โหนดใช้ชื่อโดเมนเดียวกับเซิร์ฟเวอร์โหนด
-n netmask	ค่าเน็ตมาสก์ของไคลเอนต์โหนด
-g gateway	เกตเวย์แอดเดรสสำหรับไคลเอนต์, ถ้าไม่ได้ระบุออพชั่นนี้ จะใช้แอดเดรสของเซิร์ฟเวอร์โหนดแทน
-N Network_address	ค่าเน็ตเวิร์กแอดเดรสของไคลเอนต์
-b broadcast	ค่าบรอดคาสท์แอดเดรสของไคลเอนต์
-D interface	อินเตอร์เฟสเช่น eth0, eth1 สำหรับใช้ค่าคอนฟิกของมันปรับแต่งเน็ตมาสก์, บรอดคาสท์, และเกตเวย์แอดเดรส

ตารางที่ 5-2 ออพชั่นของสคริปต์ adcn

ตัวอย่างการใช้งานสคริปต์เช่น

```
# ./adcn -f  -i 161.246.5.158 -l -c node1 -d ce.kmitl.ac.th -s 161.246.5.157 -D eth0
```

คำสั่งข้างบนเป็นการรันสคริปต์ adcn เพื่อทำการเพิ่มไคลเอนต์โหนดเข้าไปในคลัสเตอร์ โดยไม่สนใจเวอร์ชั่นของลินุกซ์ที่ใช้อยู่ และกำหนดให้ไคลเอนต์โหนดที่ไอพีแอดเดรสเป็น 161.246.5.158 รอฟังสัญญาณ RARP request จากอินเตอร์เฟส eth0 โดยไคลเอนต์โหนดมีชื่อเป็น node1, ใช้โดเมน ce.kmitl.ac.th และใช้แอดเดรสเซิร์ฟเวอร์ 161.246.5.157

สคริปต์นี้จะใช้ไดเรกทอรี /tftpboot/Template ที่ถูกสร้างขึ้นโดย sdct สคริปต์ เพื่อใช้เป็นระบบไฟล์ NFS ของไคลเอนต์โหนด โดย /tftpboot/Template จะเสมือนเป็นแม่พิมพ์ของระบบไฟล์ และสคริปต์ adcn นี้จะนำระบบไฟล์แม่พิมพ์ไปทำสำเนาขึ้นมาใหม่ให้กับไคลเอนต์และปรับแต่งค่าคอนฟิกต่างๆให้เหมาะกับไคลเอนต์โหนดต่อไป หากไม่พบไดเรกทอรี /tftpboot/Template ให้ทำการเรียกสคริปต์ sdct ดังแสดงในหัวข้อ 5.3.3

***หมายเหตุ

การเรียกใช้สคริปต์ adcn โดยใช้ออพชั่น -l นั้น สคริปต์จะทำการรอสัญญาณ RARP request, เราต้องนำแผ่นบู๊ตสำหรับไคลเอนต์ที่ได้สร้างขึ้นในขั้นตอนที่ 5.3.4 มาใส่ที่เครื่องไคลเอนต์แล้วทำการบู๊ตเคอร์เนลที่อยู่ในแผ่นบู๊ตจะทำการส่งสัญญาณ RARP request ซึ่งจะประกอบด้วยแมคแอดเดรสของอินเตอร์เฟสการ์ดของเครื่องไคลเอนต์ออกไป แล้วสคริปต์ adcn จึงจะทำงานต่อไปได้ ดังนั้นขณะทำการรันสคริปต์นี้มีข้อควรระวังคือ

- จะต้องไม่มีเครื่องอื่นใดในเครือข่าย พยายามจะบู๊ตจากเครือข่ายและส่งสัญญาณ RARP request ออกมา
- สคริปต์นี้จะทำการจดจำแมคแอดเดรสของอินเตอร์เฟสการ์ดของไคลเอนต์ไว้ใน RARP table ดังนั้นหากมีการเปลี่ยนแปลงอินเตอร์เฟสการ์ดในเครื่องไคลเอนต์ใดๆ ต้องทำการลบระบบไฟล์ของไคลเอนต์ทิ้งแล้วทำการเพิ่มไคลเอนต์เข้าไปใหม่โดยเรียกสคริปต์ adcn นี้อีกที หรือเข้าไปเปลี่ยนแปลง RARP table โดยตรง โดยสามารถดูรายการใน RARP table โดยใช้คำสั่ง

```
# rarp –a
```

การลบรายการใน RARP table ใช้คำสั่ง

```
# rarp –d hostname
```

การเพิ่มรายการใน RARP table ใช้คำสั่ง

```
# rarp -s hostname hw_address
```

และหากสงสัยสามารถดูวิธีการใช้งานได้โดยใช้คำสั่ง man rarp, ตัวอย่างข้างล่างเป็น RARP table ของคลัสเตอร์ที่เราสร้างขึ้น

IP address	HW type	HW address
161.246.5.158	10Mbps Ethernet	02:60:8c:6c:90:a5

5.3.6 ปรับแต่งเซิร์ฟเวอร์

5.3.6.1 /etc/hosts

ไฟล์นี้จะเก็บรายชื่อเครื่องต่างๆและไอพีแอดเดรสของเครื่องนั้นๆไว้ ในกรณีที่เราจะอ้างถึงเครื่องใดๆในเครือข่าย เราเพียงแต่ระบุชื่อเครื่องนั้นๆเพียงอย่างเดียว แล้วระบบจะทำการหาไอพีแอดเดรสที่สอดคล้องกับชื่อเครื่องนั้นได้จากไฟล์ /etc/hosts นี้ แต่ถ้าหาไม่พบระบบก็จะไปหาจาก DNS เซิร์ฟเวอร์อีกทีหนึ่ง แต่เพื่อความรวดเร็วเราควรใส่ชื่อเครื่องและไอพีแอดเดรสของทุกโหนดในคลัสเตอร์ไว้เลย เพื่อจะได้ไม่เสียเวลาไปดึงข้อมูลจาก DNS เซิร์ฟเวอร์อีกต่อหนึ่ง ปกติแล้วไฟล์ /etc/hosts จะถูกทำลิงค์ไปยังระบบไฟล์ของทุกๆโหนดโดยสคริปต์ sdct เพื่อให้ทุกๆโหนดใช้ข้อมูลร่วมกัน และสคริปต์ adcn จะทำการเพิ่มรายการชื่อเครื่องของไคลเอนต์ให้ทุกครั้งที่เรียกโดยอัตโนมัติ แต่เราสามารถทำการแก้ไขไฟล์นี้ได้เองโดยใช้โปรแกรมเอดิเตอร์ทั่วๆไป, ต่อไปเป็นตัวอย่างไฟล์ /etc/hosts ของคลัสเตอร์ที่เราสร้างขึ้น

```
127.0.0.1       localhost.localdomain   localhost
161.246.5.157   network07.ce.kmitl.ac.th  node0
161.246.5.158   network08.ce.kmitl.ac.th  node1
```

5.3.6.2 /etc/hosts.equiv

เพื่อที่จะให้โหนดใดๆสามารถรันรีโมตเชลล์ (rsh) ไปยังโหนดอื่นๆในคลัสเตอร์ได้ สำหรับทุกๆผู้ใช้งาน (user) โดยที่ไม่ต้องทำการรอใส่พาสเวิร์ด เราต้องทำการใส่รายชื่อของทุกๆโหนดไว้ที่ไฟล์ /etc/hosts.equiv ดังตัวอย่างข้างล่างนี้

```
node0
node1
```

5.3.6.3 .rhosts

ไฟล์ .rhosts นี้จะคล้ายๆกับไฟล์ /etc/hosts.equiv คือจะเก็บรายชื่อเครื่องต่างๆในคลัสเตอร์ไว้ เพื่อให้เครื่องเหล่านั้นสามารถรันรีโมตเชลล์บนเครื่องอื่นๆได้โดยไม่ต้องมีการรอใส่พาสเวิร์ด เพียงแต่ไฟล์ .rhosts นี้จะใช้ได้เพียงผู้ใช้งาน (user) เดียวเท่านั้น เราจะต้องทำการสร้างไฟล์นี้ในโฮมไดเรกทอรี (home directory) ของทุกๆผู้ใช้งาน (user) ในขณะที่ไฟล์ /etc/hosts.equiv เพียงไฟล์เดียวนั้นนั้นสามารถใช้ได้กับทุกๆผู้ใช้งานในโหนด และไฟล์ /etc/hosts.equiv นี้ ยังถูกลิงค์ด้วยสคริปต์ sdct และ adcn ทำให้ทุกๆโหนดใช้งานไฟล์นี้ร่วมกัน ดังนั้นการเปลี่ยนแปลงไฟล์นี้เพียงไฟล์เดียวจึงเหมือนกับเป็นการอนุญาติให้สามารถรันรีโมตเชลล์ได้ทั้งคลัสเตอร์

5.3.6.4 การรันคำสั่ง rlogin ด้วย root

เพื่อที่จะให้ root สามารถรันคำสั่ง rlogin ไปยังทุกๆโหนดในคลัสเตอร์ได้ เราต้องสร้างไฟล์ .rhosts ซึ่งประกอบด้วยรายชื่อโหนดในคลัสเตอร์นั้นไว้ในรูทไดเรกทอรี (root directory) ของทุกๆโหนดในคลัสเตอร์ด้วย และเพื่อความปลอดภัยต้องไม่ให้ผู้ใช้งานอื่นมาเปลี่ยนแปลงไฟล์นี้โดยใช้คำสั่ง

```
# chmod go-rwx .rhosts
```

และต้องทำการสลับสองบรรทัดแรกในไฟล์ /etc/pam.d/rlogin ดังตัวอย่างข้างล่างต่อไปนี้

<u>ไฟล์ /etc/pam.d/rlogin ต้นฉบับ</u>

```
auth       required     /lib/security/pam_securetty.so
auth       sufficient   /lib/security/pam_rhosts_auth.so
auth       required     /lib/security/pam_pwdb.so shadow nullok
auth       required     /lib/security/pam_nologin.so
account    required     /lib/security/pam_pwdb.so
password   required     /lib/security/pam_cracklib.so
password   required     /lib/security/pam_pwdb.so shadow nullok use_authtok
session    required     /lib/security/pam_pwdb.so
```

<u>เมื่อทำการสลับสองบรรทัดแรกแล้ว</u>

```
auth       sufficient   /lib/security/pam_rhosts_auth.so
auth       required     /lib/security/pam_securetty.so
auth       required     /lib/security/pam_pwdb.so shadow nullok
auth       required     /lib/security/pam_nologin.so
account    required     /lib/security/pam_pwdb.so
password   required     /lib/security/pam_cracklib.so
```

```
password   required    /lib/security/pam_pwdb.so shadow nullok use_authtok
session    required    /lib/security/pam_pwdb.so
```

5.3.7 ติดตั้งระบบรักษาความปลอดภัย

เนื่องจากระบบคลัสเตอร์นั้นจำเป็นต้องเปิดให้แต่ละโหนดสามารถเข้าถึงโหนดอื่นๆในคลัสเตอร์ได้ จึงต้องคลายข้อจำกัดต่างๆทางด้านความปลอดภัยลง เช่น การอนุญาติให้ใช้คำสั่ง rlogin โดยไม่ต้องใส่พาสเวิร์ด ดังนั้นจึงอาจมีช่องโหว่ให้ผู้ไม่ประสงค์ดีแฮกเข้ามาในระบบ เราจึงต้องสร้างระบบรักษาความปลอดภัยที่อนุญาติให้เฉพาะเครื่องระบบเท่านั้นที่เข้าถึงโหนดต่างๆในระบบ ในขณะที่กันระบบออกจากโลกภายนอกไม่ให้มีโอกาสเข้าถึงได้ซึ่งมีวิธีการต่างๆดังนี้

5.3.7.1 ทีซีพีแวรพเปอร์ (TCP wrapper)

ทีซีพีแวรพเปอร์หรือทีซีพีดีเดมอน (tcpd daemon) เป็นเดมอนตัวที่ใช้กรองแพ็กเก็ต (packet) ที่ส่งผ่านกันในเครือข่ายชั้นแรก ซึ่งเราสามารถกำหนดได้จากไฟล์ 3 ตัวคือ /etc/hosts.allow, /etc/hosts.deny และ /etc/inetd.conf แต่ละไฟล์มีรูปแบบและรายละเอียดดังนี้

5.3.7.1.1 /etc/hosts.allow

ไฟล์ /etc/hosts.allow นี้จะระบุไอพีแอดเดรสของเครื่องที่สามารถเข้ามาใช้บริการ และบริการต่างๆที่เครื่องนั้นสามารถเข้ามาใช้ได้ ตัวอย่างต่อไปนี้แสดงการอนุญาติให้เครื่องใดก็ได้ที่มีไอพีแอดเดรส 161.246.5.157 และ 161.246.5.158 สามารถเข้ามาใช้บริการได้ทุกพอร์ต และสามารถใช้เทลเน็ต (telnet) ได้จากเครื่องที่มีไอพีแอดเดรส 161.246.5.229 ดังนี้

```
#
# hosts.allow   This file describes the names of the hosts which are
#               allowed to use the local INET services, as decided
#               by the '/usr/sbin/tcpd' server.
#

# we fully trust ourself and all the other nodes within the cluster

ALL : localhost, 161.246.5.157, 161.246.5.158

In.telnetd : 161.246.5.229
```

5.3.7.1.2 /etc/hosts.deny

ระบบจะทำการหาชื่อเครื่องหรือไอพีแอดเดรสที่อยู่ในไฟล์ /etc/hosts.allow ถ้าพบแสดงว่าเครื่องนั้นสามารถเข้ามาใช้บริการได้ตามที่ระบุไว้ในไฟล์ แต่ถ้าไม่พบระบบจะเข้ามาหาชื่อเครื่องหรือไอพีแอดเดรสในไฟล์ /etc/hosts.deny นี้ หากตรงกับในไฟล์นี้เครื่องนั้นก็ไม่สามารถใช้บริการต่างๆตามที่ได้ระบุไว้ในไฟล์นี้ได้ เพื่อความปลอดภัยหากไม่ได้ระบุไว้ในไฟล์ /etc/hosts.allow เราจะปิดทุดอย่างไม่ให้เครื่อง

นอกเหนือจากนี้เข้ามาใช้บริการได้ และเราจะเขียนสคริปต์เพื่อให้ทุกครั้งที่มีการปฏิเสธการขอใช้บริการจะมีการส่งอีเมลแอดเดรสไปยังผู้ดูแลระบบด้วย ดังตัวอย่างข้างล่างนี้

```
ALL: ALL: spawn ( \
Echo -e "\n\
TCP Wrappers\:  Connection Refused\n\
By\:                    $(uname -n)\n\
Process\:               %d (pid %p)\n\
User\:                  %u\n\
Host\:                  %c\n\
Date\:                  $(date)\n\
" | /bin/mail -s "From tcpd@$(uname -n).  %u@%h -> %d." root)
```

เมื่อเครื่องภายนอกพยายามจะติดต่อเข้ามายังคลัสเตอร์และไม่พบรายชื่อเครื่องนั้นในไฟล์ /etc/hosts.allow การติดต่อจะถูกปิดลงและจะมีอีเมลแจ้งเตือนไปยังผู้ดูแลระบบซึ่งมีลักษณะดังนี้

```
From root  Fri Apr 16 23:33:50 1999
Return-Path: <root>
Received: (from root@localhost)
        By network07.ce.kmitl.ac.th (8.8.7/8.8.7) id XAA19278
        for root; Fri, 16 Apr 1999 23:33:50 +1000
Date: Fri, 16 Apr 1999 23:33:50 +1000
From: network07 Admin <root@network07.ce.kmitl.ac.th>
Message-Id: <199904161333.XAA19278@network07.ce.kmitl.ac.th>
To: root@network07.ce.kmitl.ac.th
Subject: From tcpd@network07.ce.kmitl.ac.th
jacek@lamport.comp.usq.edu.au -> in.telnetd.
Status: O


TCP Wrappers:  Connection Refused
By:                    network07.ce.kmitl.ac.th
Process:               in.telnetd (pid 19270)
User:                  jacek
Host:                  jacek@lamport.comp.usq.edu.au
Date:                  Fri Apr 16 23:33:50 EST 1999
```

5.3.7.1.3 /etc/inetd.conf

ไฟล์นี้จะระบุถึงบริการต่างๆที่เปิดให้ใช้ ซึ่งเดมอนหลายๆตัวก็ถูกเปิดให้ใช้งานจาก inetd ซึ่งสามารถปรับแต่งค่าต่างๆได้จากไฟล์ inetd.conf นี้, บริการไหนไม่จำเป็นต้องใช้ หรือเสี่ยงต่อการรุกล้ำเข้ามาในระบบให้ทำการปิด, การปรับแต่งไฟล์สามารถทำได้ดังนี้

```
Shell    stream  tcp     nowait  root    /usr/sbin/tcpd  in.rshd
#login   stream  tcp     nowait  root    /usr/sbin/tcpd  in.rlogind
#exec    stream  tcp     nowait  root    /usr/sbin/tcpd  in.rexecd
#talk    dgram   udp     wait    root    /usr/sbin/tcpd  in.talkd
#ntalk   dgram   udp     wait    root    /usr/sbin/tcpd  in.ntalkd
```

จากตัวอย่างข้างบนเป็นการปิดบริการ login, exec, talk และ ntalk โดยการเข้าไปเปลี่ยนแปลงไฟล์ /etc/inetd.conf นี้เราจะต้องทำการเริ่มต้น inetd ใหม่โดยใช้คำสั่ง

```
# killall –HUP inetd
```

เราสามารถทำการตรวจสอบบริการที่เปิดอยู่ได้โดยใช้คำสั่ง

```
# netstat -a | grep "LISTEN" | grep -v "^unix"
```

5.3.7.2 ปิดบริการที่เปิดขึ้นมาจากสคริปต์ rc

บริการหลายๆตัวถูกเปิดขึ้นโดยสคริปต์ rc เช่น เว็บเซิร์ฟเวอร์และแซมบ้าเซิร์ฟเวอร์เป็นต้น เราต้องไปตัดการทำงานของสคริปต์ rc ดังคำสั่งดังต่อไปนี้

```
# rm -f /etc/rc.d/rc3.d/S*httpd
# rm -f /etc/rc.d/rc5.d/S*httpd
# rm -f /etc/rc.d/rc3.d/S*smb
# rm -f /etc/rc.d/rc5.d/S*smb
# rm -f /etc/rc.d/rc3.d/S*sendmail
# rm -f /etc/rc.d/rc5.d/S*sendmail
```

5.3.7.3 ไฟร์วอล (firewall)

การที่เราจะปิดกั้นแพ็กเก็ตที่มาจากไอพีแอดเดรสโดยการระบุพอร์ตที่จะทำการปิดกั้น จะทำให้มีความยืดหยุ่นมากกว่าการใช้ทีซีพีแวรพเปอร์ ตัวอย่างของสคริปต์ไฟร์วอลสามารถดูได้จากภาคผนวก, ตัวอย่างข้างล่างต่อไปนี้จะเป็นการสั่งทำงานสคริปต์ไฟร์วอลโดยอัตโนมัติทุกครั้งที่ทำการบูต

```
# cp /home/jacek/firewall /etc/rc.d/init.d
# chmod u+rx firewall
# ln –s /etc/rc.d/init.d/firewall /etc/rc.d/rc3.d/S05firewall
# ln –s /etc/rc.d/init.d/firewall /etc/rc.d/rc5.d/S05firewall
```

5.3.8 ติดตั้งเมสเซจพาสซิ่งไลบรารี่

เบียวูล์ฟคลัสเตอร์ประมวลผลโปรแกรมเชิงขนานโดยใช้ไลบรารี่เมสเซจพาสซิ่ง เช่น MPI และ PVM เป็นส่วนประกอบ และโดยทั่วๆไปแล้วเรดแฮทลินุกซ์ (Red Hat Linux ก็จะมีแพ็กเก็จนี้มาให้เลือกตอนติดตั้งอยู่แล้วด้วย แต่อย่างไรก็ตามเราสามารถหาซอฟต์แวร์เหล่านี้มาติดตั้งเองได้ดังนี้

5.3.8.1 MPI

5.3.8.1.1 MPICH

MPICH เป็นเวอร์ชั่นหนึ่งของ MPI ที่มีประสิทธิภาพมาก สามารถหาได้จาก

http://www-unix.mcs.anl.gov/mpi/mpich

5.3.8.1.2 LAM-MPI

LAM-MPI สามารถหาติดตั้งได้จาก

http://www.mpi.nd.edu/lam

การรัน LAM นั้นจะต้องทำการเริ่มต้น LAM daemon บนทุกๆโหนด โดยใช้คำสั่งต่อไปนี้

```
# lamboot -v lamhosts
```

โดย lamhosts เป็นไฟล์รายการที่เป็นชื่อโหนดที่เราต้องการให้รัน LAM daemon และเราสามารถตรวจสอบการทำงานของเดมอนว่ายังทำงานอยู่หรือไม่โดยใช้คำสั่ง

```
# ps auxw | grep lam
```

หรือสามารถตรวจสอบงานที่กำลังรันอยู่ด้วยคำสั่ง

```
# mpitask
```

และหยุดการทำงานของเดมอนด้วยคำสั่ง

```
# wipe lamhosts
```

เราสามารถสั่งรันงาน MPI ได้ด้วยคำสั่งดังเช่น

```
# mpirun -O -c 2 -s h -c2c program
```

- O หมายถึงเป็นโฮโมจีเนียสคือทุกๆโหนดเหมือนกันหมด
- c หมายถึงจำนวนงานที่จะแบ่งออกไปรันบนเครื่องต่างๆ (ในที่นี้คือ 2)
- s หมายถึงที่อยู่ของงานที่จะรัน h หมายถึง hosts หรือเครื่องที่เปิด LAM ขึ้นมา
- c2c หมายถึงใช้ซ็อกเก็ตแบบไคลเอนต์ไปยังไคลเอนต์ ซึ่งวิธีนี้จะเร็วและมีประสอทธิภาพ แต่จะไม่สามารถเข้าสู่โหมดดีบักได้

5.3.8.2 PVM

สามารถหาแพ็กเกจของ PVM เพิ่มเติมได้ที่

http://www.epm.ornl.gov/pvm/

และสามารถหาข้อมูลเพิ่มเติมได้ที่

http://netlib.org/pvm3/book/pvm-book.html

บทที่ 6
การสร้างส่วนติดต่อกับผู้ใช้งานบนเอ็กซ์วินโดว์

6.1 เอ็กซ์วินโดว์ (X Window)

6.1.1 เอ็กซ์วินโดว์คืออะไร

เอ็กซ์วินโดว์ (X Window) หรือบางทีเรียกย่อๆว่าเอ็กซ์ (X) เป็นโปรแกรมที่สร้างขึ้นมาเพื่อครอบบนระบบปฏิบัติการอีกทีหนึ่ง และทำให้สามารถใช้งานคอมพิวเตอร์โดยผ่านทางการแสดงผลที่เป็นรูปภาพแบบกราฟิกร่วมกับเมาส์ โดยมีการแบ่งพื้นที่แสดงผลออกเป็นหลายหน้าต่างย่อยๆ หรือเรียกว่า window นั่นเอง ซึ่งเป็นระบบอินเตอร์เฟสที่เรียกว่า GUI (Graphic User Interface)

6.1.2 ประวัติความเป็นมาของเอ็กซ์วินโดว์

การพัฒนาเรื่อง GUI บนยูนิกซ์มีประวัติมายาวนานตั้งแต่ปี ค.ศ. 1987 โดยสถาบันเทคโนโลยีแมสซาชูเซท (Massachusetts Institute of Technology, MIT) ได้ประกาศ snapshot แรกของเวอร์ชันที่ 11 ของระบบเอ็กซ์วินโดว์ หรือที่รู้จักกันในนาม X11 ซึ่ง X11 ถือว่าเป็นเทคโนโลยีสำคัญมากที่สุดเทคโนโลยีหนึ่งในทศวรรษที่ 1990

X เป็นโครงการที่ถูกพัฒนาร่วมกันระหว่างโปรเจคอะธีนาของ MIT และบริษัทดิจิตอลอิควิปเมนต์คอปอเรชัน (DEC) รวมทั้งได้รับการสนับสนุนจากบริษัทอื่นๆอีกมากมาย โครงการนี้อยู่ภายใต้การดูแลของ Robert Scheifler และเพื่อนร่วมงานของเขาจาก MIT และโครงการนี้ได้รับอิทธิพลบางส่วนมาจากโครงการ W ซึ่งเป็นแพ็กเกจสำหรับวินโดว์ที่พัฒนาโดย Paul Asente จากมหาวิทยาลัยสแตนฟอร์ด

6.1.3 ส่วนประกอบของเอ็กซ์วินโดว์

ระบบเอ็กซ์วินโดว์เป็นระบบที่พัฒนาขึ้นโดยไม่ขึ้นอยู่กับสถาปัตยกรรมแบบใดเลย รวมทั้งมีพื้นฐานอยู่บนระบบเครือข่าย นั่นคือมีลักษณะการทำงานแบบไคลเอนต์เซิร์ฟเวอร์ โปรแกรมหลักของเอ็กซ์วินโดว์หรือส่วนที่เป็นเซิร์ฟเวอร์จะเรียกว่าเอ็กซ์เซิร์ฟเวอร์ (X server) ซึ่งจะทำหน้าที่ดูแลการแสดงผลบนจอภาพ จับตำแหน่งของเมาส์ รวมทั้งรับข้อมูลที่ป้อนผ่านทางคีย์บอร์ด ขณะที่เอ็กซ์ไคลเอนต์ (X client) จะเป็นโปรแกรมใช้งานทั่วๆไป เมื่อต้องการแสดงผลก็จะส่งข้อมูลไปให้เอ็กซ์เซิร์ฟเวอร์ ซึ่งการติดต่อกันระหว่างเอ็กซ์เซิร์ฟเวอร์และเอ็กซ์ไคลเอนต์นี้จะใช้โปรโตคอลแบบเอ็กซ์โปรโตคอล (X Protocal)

เอ็กซ์เซิร์ฟเวอร์เป็นเสมือนรากฐานของระบบ แต่ระบบเอ็กซ์วินโดว์ที่มีเฉพาะเอ็กซ์เซิร์ฟเวอร์เพียงอย่างเดียวจะไม่สามารถใช้ประโยชน์อะไรได้ จะต้องมีแอพพลิเคชันอื่นๆ ซึ่งแอพพลิเคชันเหล่านี้จะถูกจัดเป็นเอ็กซ์ไคลเอนต์ทั้งหมด โดยที่เอ็กซ์ไคลเอนต์เหล่านี้สามารถทำงานข้ามระบบเครือข่ายได้อีกด้วย ผลก็คือระหว่างตัวไคลเอนต์กับเซิร์ฟเวอร์อาจอยู่บนเครื่องเดียวกัน หรืออยู่คนละเครื่องก็ได้โดยที่

เครื่องเหล่านั้นจะต้องเชื่อมต่อกันเป็นระบบเครือข่ายบนโปรโตคอลแบบ TCP/IP นั่นหมายความว่าเราสามารถรันแอพพลิเคชันบนเครื่องอื่นๆและกำหนดให้มาแสดงผลบนเครื่องเราได้

6.1.4 การปรับแต่งเอ็กซ์วินโดว์

ปกติแล้วเราสามารถติดตั้งเอ็กซ์วินโดว์ในขณะติดตั้งระบบปฏิบัติการลินุกซ์ได้เลย แต่ถ้าหากเราไม่ได้ทำการติดตั้งเอ็กซ์วินโดว์แต่แรก หรืออยากเปลี่ยนแปลงสิ่งที่ได้เซ็ตอัพไว้แต่แรก เราสามารถปรับแต่งเอ็กซ์วินโดว์ใหม่ได้ โดยการปรับแต่งนี้จะหมายถึงการกำหนดให้เอ็กซ์เซิร์ฟเวอร์สามารถใช้งานร่วมกับการ์ดแสดงผล, จอมอนิเตอร์ รวมทั้งอุปกรณ์อินพุตเช่น เมาส์, คีย์บอร์ด สำหรับเอ็กซ์เซิร์ฟเวอร์ที่ลินุกซ์ใช้อยู่คือ Xfree86 เวอร์ชัน 3.3 ซึ่งสามารถใช้งานได้กับอุปกรณ์ที่มีอยู่ในท้องตลาดเกือบทุกยี่ห้ออยู่แล้ว แต่อาจมีอุปกรณ์บางตัวที่ไม่สามารถตรวจสอบได้โดยอัตโนมัติ ทำให้เราต้องทำการปรับแต่งด้วยตนเอง

โปรแกรมสำหรับการปรับแต่งเอ็กซ์วินโดว์ในเรดแฮท (Red Hat) นี้มีให้เลือกใช้สองโปรแกรม คือ Xconfigurator และ xf86config ซึ่งทั้งสองโปรแกรมจะทำหน้าที่สองอย่างคือ

- สร้างลิงค์ไฟล์ ซึ่งเป็นการบอกให้ทราบว่าลินุกซ์ของเราใช้โปรแกรมเอ็กซ์เซิร์ฟเวอร์อะไร ไฟล์นี้จะอยู่ที่ /etc/X11/X
- รวบรวมข้อมูลทุกอย่างที่ต้องใช้ในงานระบบกราฟิกไม่ว่าจะเป็น ข้อมูลของการ์ดแสดงผล, มอนิเตอร์, เมาส์, คีย์บอร์ด หรืออุปกรณ์อินพุตตัวอื่นๆ ซึ่งข้อมูลเหล่านี้จะถูกเก็บลงในไฟล์ /etc/X11/XF86Config

การปรับแต่งด้วย Xconfigurator เรียกโดยใช้คำสั่งต่อไปนี้

```
# Xconfigurator
```

6.2 ทิเคิล (TCL)

ทิเคิล (TCL Tool Command Language) เป็นภาษาที่ใช้ควบคู่กับภาษาทีเค (TK) เพื่อสร้างแอพพลิเคชั่นสำหรับใช้งานบนเอ็กซ์วินโดว์ ทิเคิลมีลักษณะคล้ายภาษาเชลล์สคริปต์ทั่วๆไป อีกทั้งยังมีความสามารถในการควบคุมการทำงานของโปรแกรมเช่น การวน (loop) เงื่อนไข (conditional command) การจัดการอีเวนต์ (Event) การจัดการลิสต์และตัวแปร นอกจากนั้นทิเคิลยังสามารถใช้ร่วมกับภาษาโปรแกรมมิ่งอื่นๆได้ด้วย เช่น C หรือ TK เป็นต้น

6.2.1 การเริ่มต้นใช้ภาษาทิเคิล

ทิคเคิลมีการทำงานคล้ายภาษาเชลล์สคริปต์ทั่วๆไป คือจะเริ่มทำงานได้ก็ต่อเมื่อต้องเข้าเชลล์ของทิเคิลก่อนแล้วจึงพิมพ์คำสั่งลงไปในเชลล์ทีละบรรทัด สำหรับทิเคิลนั้นจะใช้เชลล์ที่เรียกว่า ทีซีเชลล์ (tclsh) โดยพิมพ์คำสั่งดังนี้

```
# tclsh
```

```
% set A 10
% set B “xxx”
% exec echo $A
10
% exec echo $B
xxx
```

โดยเครื่องหมาย % จะเป็นพร้อมท์ สำหรับรอรับคำสั่ง ในตัวอย่างข้างต้นเป็นการเรียกใช้ทีซีเชลล์ และกำหนดให้ ตัวแปร A มีค่าเท่ากับ 10, กำหนดให้ตัวแปร B มีค่าเท่ากับ string xxx และทำการแสดงค่า A และ B ออกมา

หรืออีกแนวทางหนึ่งก็คือเขียนสคริปต์ของภาษาทิเคิลเก็บไว้ในไฟล์ แล้วเรียกใช้ไฟล์สคริปต์นั้นผ่านเชลล์อีกชั้นหนึ่ง ดังตัวอย่างต่อไปนี้

ตัวอย่างไฟล์สคริปต์ภาษาทิคเคิล สมมุติชื่อ myfile.tcl

```
# !/usr/bin/tclsh
set A 10
set B “xxx”
exec echo $A
exec echo $B
```

เรียกไฟล์ขึ้นมาทำงานโดยใช้คำสั่งต่อไปนี้

```
# ./myfile.tcl
```

6.2.2 คำสั่งเบื้องต้นของทิเคิล

คำสั่งเบื้องต้นของทิเคิลที่ใช้บ่อยๆ และมีความจำเป็นมากมีดังต่อไปนี้

exec ใช้สำหรับประมวลผลคำสั่งเชลล์อื่นๆ โดยค่าดีฟอลต์จะเป็นเชลล์ก่อนหน้าที่จะเข้ามาสู่ทีซีเชลล์ เช่น ถ้าเราจะใช้คำสั่ง ping ใน bash เชลล์ เราสามารถทำได้โดยการเรียกดังนี้

```
exec command
exec ping 161.246.10.21 e.g.
exec echo hello world e.g.
```

global ใช้ประกาศค่าตัวแปรให้เป็นแบบโกลบอล คือใช้ได้ทั่วถึงทั้งโปรแกรมและทุกๆโพรซีเดอร์ย่อย เช่น

```
global varable

global A e.g.
gloabal B e.g.
```

if เป็นคำสั่งที่ใช้ตรวจสอบเงื่อนไข ถ้าตรงตามเงื่อนไขจึงจะทำคำสั่งที่กำหนดไว้ เช่น

```
if condition command

if {A == 10} {
exec echo A is equal to Ten
}

if {B != 'xxx'} {
exec echo B isn't equal to xxx
}
```

incr เป็นคำสั่งสำหรับเพิ่มหรือลดค่าของตัวแปร เช่น

```
incr variable number

incr A 20
incr A 1
incr A – 3
```

list เป็นคำสั่งที่ใช้จัดการเกี่ยวกับลิสต์ ซึ่งประกอบไปด้วยคำสั่งย่อยต่างๆมากมาย เช่น

```
lindex รีเทิร์นค่า ณ ตำแหน่งใดๆในลิสต์ออกมา
list สร้างลิสต์ขึ้นมาใหม่
lsearch ค้นหาคำที่อยู่ในลิสต์ และรีเทิร์นค่าตำแหน่งที่พบออกมา
lrange รีเทิร์นช่วงใดช่วงหนึ่งของลิสต์ออกมา
```

string เป็นคำสั่งที่ใช้จัดการสตริงซึ่งประกอบด้วยคำสั่งย่อยๆดังนี้

```
string length หาค่าความยาวของสตริง
string trimleft ลบอักขระทางซ้ายของสตริง
```

```
string trimright ลบอักขระทางขวาของสตริง
string search ค้นหาคำในสตริง
string range ตัดเฉพาะบางส่วนของสตริง
```

set เป็นคำสั่งสำหรับทำการกำหนดค่าให้กับตัวแปร

```
set variable value

set A 10 e.g.
set B [exec cat /etc/motd] e.g.
```

while เป็นคำสั่งสำหรับวนลูป เช่น

```
while {i != 10} {
exec echo $i
incr i
}
```

6.3 ทีเค (TK)

ทีเคเป็นภาษาสำหรับสร้างวินโดว์อินเตอร์เฟส (window interface) แต่โดยปกติแล้วในยูนิกซ์เราจะเรียกว่า วิดเก็ท (widget) แทนที่จะเรียกว่าวินโดว์ ทีเคจะเป็นภาษาเชลล์สคริปต์ตัวหนึ่งที่มีความสามารถในการสร้างวิดเก็ทต่างๆ ได้อย่างง่ายดาย ตัวอย่างวิดเก็ทที่ใช้บ่อยเช่น เฟรม(frame), ปุ่ม(button), รูปภาพ (image), แถบเลื่อน (scroll bar), ข้อความ (message), ลิสต์บ๊อกซ์ (listbox) เป็นต้น

ทีเคนั้นจะมีลักษณะคล้ายทิเคิลอยู่คือ เป็นภาษาสคริปต์เหมือนกัน โดยทิเคิลนั้นจะมีความสามารถในการความคุมโฟลว์ของโปรแกรม ในขณะที่ทีเคจะมีความสามารถในการสร้างวิดเก็ท รวมถึงสามารถตั้งอีเวนต์ต่างๆได้ และทั้งทิเคิลและทีเคนั้นสามารถใช้งานร่วมกันได้ด้วย การทำงานร่วมกันของทั้งทิเคิลและทีเคนี้ทำให้เกิดแอพพลิเคชั่นบนเอ็กซ์วินโดวต่างๆขึ้นมากมาย ซึ่งการใช้ทีเคนั้นมีวิธีการดังนี้

6.3.1 การเรียกใช้งานทีเค

การเรียกใช้งานทีเคจะคล้ายกับการเรียกใช้งานทิเคิลคือต้องเข้าเชลล์เพื่อรอรับคำสั่ง สำหรับทีเคนั้น จะเรียกว่า วินโดว์อินเตอร์เฟสเชลล์ หรือ วิช (WISH, Window Interface Shell) ซึ่งเรียกใช้งานได้ดังนี้

```
# wish
%
```

โดยพิมพ์คำสั่ง wish ที่เชลล์พร้อมท์ และเมื่อเข้าสู่ wish แล้วจะมี % ปรากฏขึ้น ซึ่งเป็นพร้อมท์ของ wish เพื่อรอรับคำสั่งต่อไปนั่นเอง

หรืออีกแนวทางหนึ่งในการเรียกใช้ทีเคคือ เขียนเป็นไฟล์สคริปต์ไว้ก่อนแล้วนำมาสั่งทำงานทั้งไฟล์ทันทีดังเช่น

ตัวอย่างไฟล์ myfile.tcl

```
# !/usr/bin/wish
toplevel .top21
message .top21.mes21 –text “ok”
place .top21.mes21 –x 10 –y 20
wm .top21 show
```

ตัวอย่างข้างต้นจะเป็นไฟล์ที่เขียนเป็นภาษาทีเค โดยทำการสร้างวินโดว์วิดเก็ทชื่อ .top21 ขึ้นมาก่อน แล้วจึงทำการสร้างข้อความคำว่า “ok” แปะลงบนพาเรนท์วิดเก็ท ซึ่งข้อความนั้นจะถือว่าเป็นวิดเก็ทลูกของ .top21 อีกทีหนึ่ง ซึ่งจะอ้างอิงกันโดยชื่อของวิดเก็ทนั้นๆด้วย เช่น .top21.mes21 จะถือว่าเป็นวิดเก็ทลูกของ .top21 โดยดีฟอลต์ ส่วน place เป็นคำสั่งสำหรับจัดวางตำแหน่งของวิดเก็ท และคำสั่ง show เป็นการแสดงวิทเก็ทชั้นบนสุด ซึ่งถือเป็นวิทเก็ทแม่ของทุกๆวิดเก็ทย่อยนั่นเอง

6.3.2 วิดเก็ท (widget)

การสร้างและจัดการวิดเก็ทเป็นส่วนที่สำคัญที่สุดของภาษาทีเค โดยภาษาทีเคสามารถสร้างวิดเก็ทต่างๆได้มากมายหลายแบบ แต่วิดเก็ทที่สำคัญๆและใช้บ่อยมีดังนี้

button ปุ่มเพื่อใช้กด และยังสามารถแทรกคำสั่งเมื่อกดปุ่มได้ด้วย

```
button name -text "Text" -command { Tcl_code... }
```

canvas สำหรับสร้างกราฟฟิกต่างๆเช่น เส้นตรง เส้นโค้ง สีเหลี่ยม วงกลม หรือรูปภาพเป็นต้น

```
canvas name

name create are x1 y1 x2 y2
```

checkbutton สำหรับทำปุ่มเช็ค

```
checkbutton name -text "Text" -variable varname \
-onvalue On -offvalue Off \
```

```
-command { Tcl_code... }
```

entry สร้างเอนทรี่วิดเก็ท เพื่อรับข้อความที่เป็นสตริง

```
entry name -width 15 –textvariable foo

name delete 0 end
name insert end string
```

frame สร้างเฟรม เพื่อแบ่งวิดเก็ทออกเป็นส่วนๆอย่างชัดเจน

```
frame name -bd 2 -relief groove
```

label สร้างป้ายข้อความ

```
label name -text "Text"
```

listbox สร้างกล่องรายการ เพื่อแสดงรายการย่อยๆในวิดเก็ท

```
listbox name.list -height 10 -selectmode single \
-yscrollcommand "name.scroll set"

scrollbar name.scroll –command { name.list yview }

name.list insert "List item"

pack name.scroll -side right -fill y

pack name.list -side left

bind name.list <Double-Button-1> { Tcl_code... }
```

message สร้างข้อความ

```
message name -text "Text" -width 2c
```

radiobutton สร้างปุ่มเรดิโอ

```
radiobutton name –text "Text" -variable varname \
-value "Print"
```

toplevel สร้างวิดเก็ทชั้นบนสุด เพื่อเป็นวิดเก็ทแม่ โดยปกติแล้วก็คือส่วนของวินโดว์ทั้งหมดนั่นเอง

```
toplevel name

wm title name "Title"

wm geometry name +X+Y

wm protocol name WM_DELETE_WINDOW { Tcl_code... }
```

บทที่ 7
โปรแกรมมอนิเตอร์ระบบประมวลผลแบบกลุ่มลินุกซ์

7.1 หลักการและการออกแบบ

เนื่องจากการโปรแกรมด้วยเอ็มพีไอนั้นมีความยากลำบากในการดีบัก อีกทั้งโครงสร้างของการโปรแกรมเชิงขนานก็ถือว่าซับซ้อน ดังนั้นการพัฒนาระบบงานขึ้นมาโดยใช้เอ็มพีไอในการแก้ไขปัญหาจึงจำเป็นต้องมีเครื่องมือต่างๆ ทั้งฮาร์ดแวร์และซอฟต์แวร์มาช่วยในการพัฒนาระบบงาน ดังที่กล่าวไว้แล้วในบทข้างต้น ความยากของการพัฒนาโปรแกรมเชิงขนานคือ อัลกอริทึมที่ซับซ้อนของโปรแกรม และการกระจายงานออกไปยังโหนดต่างๆ ทำให้ยากต่อการตรวจสอบหาข้อผิดพลาดของโปรแกรม โปรแกรมมอนิเตอร์ที่ออกแบบขึ้นนี้จึงมีวัตถุประสงค์เพื่อ

7.1.1 วัตถุประสงค์ของการออกแบบ

- เฝ้าดูการทำงานของระบบคลัสเตอร์อย่างกว้างๆ เช่น สภาพความพร้อมทำงานของระบบ, โหนดยังทำงานดีอยู่หรือไม่ มีสภาพเป็นอย่างไร ใช้ทรัพยากรณ์ไปเท่าไร เหลืออยู่เท่าไร เป็นต้น
- แยกการทำงานในส่วนของการจัดการระบบคลัสเตอร์บางส่วนออกจากผู้ใช้งาน โดยที่ผู้ใช้งานไม่จำเป็นต้องมาจัดการระบบมากนัก ผู้ใช้งานเพียงแต่สร้างและพัฒนาโปรแกรมเชิงขนาน และปล่อยให้การจัดการระบบเป็นของโปรแกรมมอนิเตอร์
- ต้องเป็นโปรแกรมที่ยืดหยุ่นสูง ติดตั้งง่าย ส่วนติดต่อกับผู้ใช้สื่อความหมาย ใช้งานง่าย
- ช่วยประหยัดเวลาของผู้ใช้งาน ในการเฝ้าติดตามดูการทำงานของเครื่องจำนวนมาก
- สามารถเก็บและบันทึกผล เพื่อใช้ในการวิเคราห์ต่อไป

7.1.2 การออกแบบ

จากจุดประสงค์ที่เรากำหนด ได้ออกแบบโปรแกรมมอนิเตอร์ขึ้นมา โดยมีฟังก์ชั่นการทำงานหลักๆอยู่ 5 อย่างคือ

- **กำหนดค่า และปรับแต่ง โหนดในคลัสเตอร์** เพื่อทำการดูข้อมูลของแต่ละโหนดในคลัสเตอร์เช่น ชื่อของโหนด, ไอพีแอดเดรส, โดเมาแนม เป็นต้น นอกจากนั้นเรายังสามารถเพิ่ม ลบ หรือทำการแก้ไขโหนดใดๆในคลัสเตอร์ได้ด้วย
- **ดูสภาพความพร้อมทำงานของระบบ** เพื่อจะดูแต่ละโหนดว่า ต่อเข้ากับคลัสเตอร์อยู่หรือไม่ สภาพความพร้อมเป็นอย่างไร หากโหนดไม่ทำงาน เราสามารถไปปรับแต่งจากเมนูข้างต้นได้ทันที
- **บันทึกและเก็บค่าการใช้งานของซีพียู** เพื่อดูการทำงานของซีพียูในแต่ละโหนดว่ามีความแตกกันอย่างไร และมีแนวโน้มไปในทิศทางใด
- **บันทึกและเก็บค่าการใช้งานของหน่วยความจำ** เพื่อนำข้อมูลการใช้จากโหนดต่างๆ นำเสนอให้ผู้ใช้งาน

- **จัดการงานหรือโปรแกรมที่เขียนด้วยเอ็มพีไอ** เพื่อนำโปรแกรมที่เขียนขึ้นโดยใช้เอ็มพีไอมารันบนคลัสเตอร์ และเฝ้าติดตามงานนั้นไปจนจบกระบวนการ รวมถึงเก็บและบันทึกผลข้อมูลการใช้ซีพียูและหน่วยความจำ เฉพาะงานนั้นๆได้

เนื่องจากโปรแกรมนี้เน้นให้ง่ายต่อการใช้งาน จึงได้มีการออกแบบส่วนติดต่อกับผู้ใช้งาน ให้ทำงานบนเอ็กซ์วินโดว์ และใช้ภาษาทิคเคิลทีเคในการพัฒนา โดยมีขั้นตอนการดำเนินงานดังในหัวข้อต่อไป

7.2 ขั้นตอนการดำเนินงาน

- ออกแบบเป้าหมายและวัตถุประสงค์ของโปรแกรมที่เราจะสร้างขึ้น
- ออกแบบฟังก์ชั่นการทำงานของโปรแกรม
- ศึกษาคำสั่งเพื่อใช้ในการสร้างฟังก์ชั่นที่ออกแบบไว้
- ออกแบบส่วนติดต่อกับผู้ใช้งาน
- ศึกษาภาษาทิคเคิลทีเค เพื่อใช้ในการสร้างส่วนติดต่อกับผู้ใช้งาน
- สร้างและทดสอบโปรแกรมมอนิเตอร์ระบบลินุกซ์คลัสเตอร์
- สรุปผล

7.3 วิธีการ

7.3.1 การกำหนดค่าและปรับแต่งโหนดในคลัสเตอร์

ในระบบคลัสเตอร์นั้นย่อมต้องประกอบไปด้วยเครื่องคอมพิวเตอร์จำนวนหลายๆเครื่อง แต่ละเครื่องจะเรียกว่าโหนด ในการจัดการคลัสเตอรนั้นเราสามารถจะลบ เพิ่ม หรือแก้ไขโหนดใดก็ได้ โดยจะมีไฟล์ที่สำคัญอยู่ด้วยกัน 2 ไฟล์คือ

/etc/hosts สำหรับเก็บรายชื่อที่จับคู่กับไอพีแอดเดรสของโหนดต่างๆไว้

/etc/hosts.equiv สำหรับเก็บรายชื่อเครื่องในคลัสเตอร์ที่อนุญาติให้ใช้รีโหมตเชลล์ระหว่างกันได้ (ถ้าไม่สามารถใช้รีโหมตเชลล์ระหว่างกันได้ก็จะไม่สามารถรันโปรแกรมเอ็มพีไอได้)

โปรแกรมจะเข้าไปอ่านและแก้ไขสองไฟล์นี้ให้โดยอัตโนมัติ สำหรับโหนดใดๆก็ได้ที่ผู้ใช้ต้องการ

7.3.2 ดูสภาพความพร้อมการทำงานของระบบ

ในส่วนนี้จะทำหน้าที่ตรวจสอบสภาพความพร้อมของระบบ โดยเริ่มจากการให้เซิร์ฟเวอร์โหนด ping ไปยังโหนดต่างๆ ในขั้นนี้หาก ping ไม่สำเร็จไม่ว่าจะด้วยเหตผลใดก็ตาม โปรแกรมจะไม่ทำงานในขั้นต่อไป เพราะถือว่าระบบไม่สามารถสื่อสารกันได้ ถ้าหากผ่านขั้นแรกแล้วต่อไปจะทำการทดสอบการรันรีโหมตเชลล์ โดยใช้คำสั่ง rsh host commandโดย host คือชื่อของโหนดปลายทาง command คือคำสั่งอะไรก็ได้ เช่น uptime เป็นต้น ถ้าหากสำเร็จโปรแกรมก็จะแสดงผลของข้อมูลทั่วๆไปของแต่ละโหนด เช่น จำนวนผู้ใช้ (user) จำนวนโปรเซส เป็นต้น

7.3.3 บันทึกและเก็บค่าการใช้งานของซีพียู

ในส่วนนี้โปรแกรมจะค่อยๆทยอยเก็บข้อมูลของแต่ละโหนดไว้ แล้วนำมาพลอตเป็นกราฟ โดยการบันทึกไว้ในไฟล์ เช่นข้อมูลซีพียูของโหนด 1 ก็จะบันทึกไว้ในไฟล์ cpunode1 เป็นต้น ส่วนคำสั่งที่ใช้ในการดูการใช้งานซีพียูคือ ใช้รีโหมตเชลล์เข้าไปยังโหนดปลายทางแล้วเก็บข้อมูลของโหนดนั้นเข้ามาไว้ โดยใช้คำสั่ง uptime อีกทีหนึ่ง ซึ่งข้อมูลที่ได้มานี้จะเป็นข้อมูลดิบ มีข้อมูลหลายอย่างปนกันมา เราต้องมาแยกออกจากกันโดยใช้คำสั่ง list ในภาษาทิคเคิลช่วยจัดการ

7.3.4 บันทึกและเก็บค่าการใช้งานของหน่วยความจำ

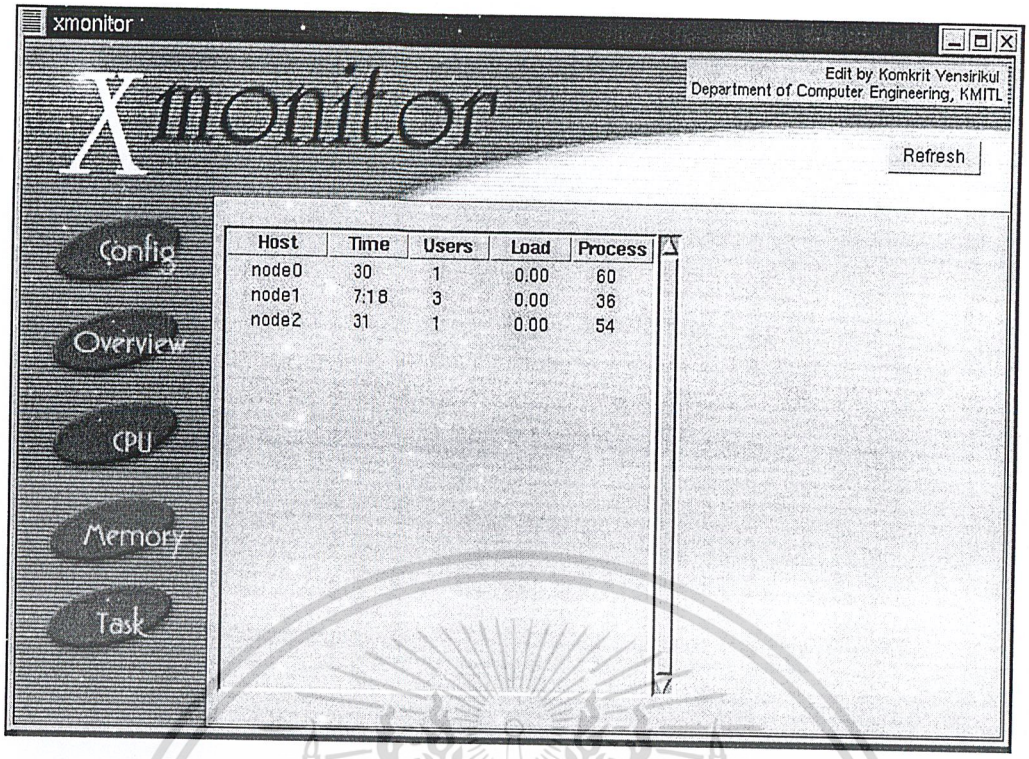
ส่วนนี้จะคล้ายๆกับการเก็บข้อมูลของซีพียู คือจะรีโหมตเชลล์เข้าไปยังโหนดปลายทาง แล้วทำการเก็บรวบรวมข้อมูลออกมา แล้วจึงทำการกรองเฉพาะส่วนที่ต้องการเท่านั้น โดยการเก็บข้อมูลของหน่วยความจำนั้นใช้คำสั่ง cat /proc/meminfo ในโหนดปลายทาง

7.3.5 การจัดการงานหรือโปรแกรมที่เขียนโดยใช้เอ็มพีไอ

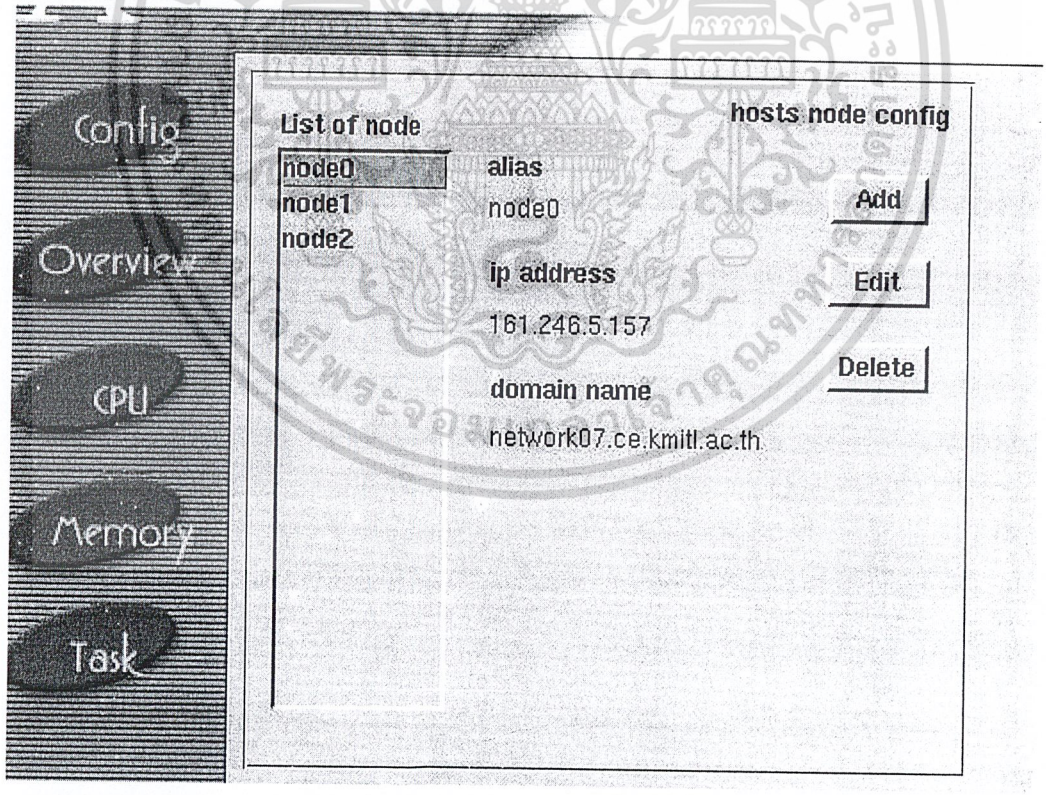
การจัดการงานจะแบ่งออกเป็นสองส่วนใหญ่ๆคือ สั่งรันงานและ เฝ้าดูการทำงานของงานที่สั่งไป การสั่งรันงานนั้นจะเริ่มต้นด้วยการเปิดแลมเดมอน (LAM daemon) แล้วสั่งรันงานนั้นด้วยคำสั่ง mpirun โดยสามารถอ่านรายละเอียดของคำสั่งได้ในบทที่ 5 ส่วนการเฝ้าติดตามงานที่เราสั่งไปนั้น ทำได้โดยตอนสั่งรันงาน จะมีการเก็บชื่อของโปรแกรมนั้นไว้ เมื่อเราเข้าไปดูโปรเซสของโปรแกรมที่เราสั่งไปแล้วนั้น จะได้ข้อมูลทั้งการใช้ซีพียูและหน่วยความจำ เมื่อเราเก็บรวบรวมข้อมูลทีละนิด จึงสามารถนำมาพล็อตเป็นกราฟได้

7.4 ผลการทดสอบ

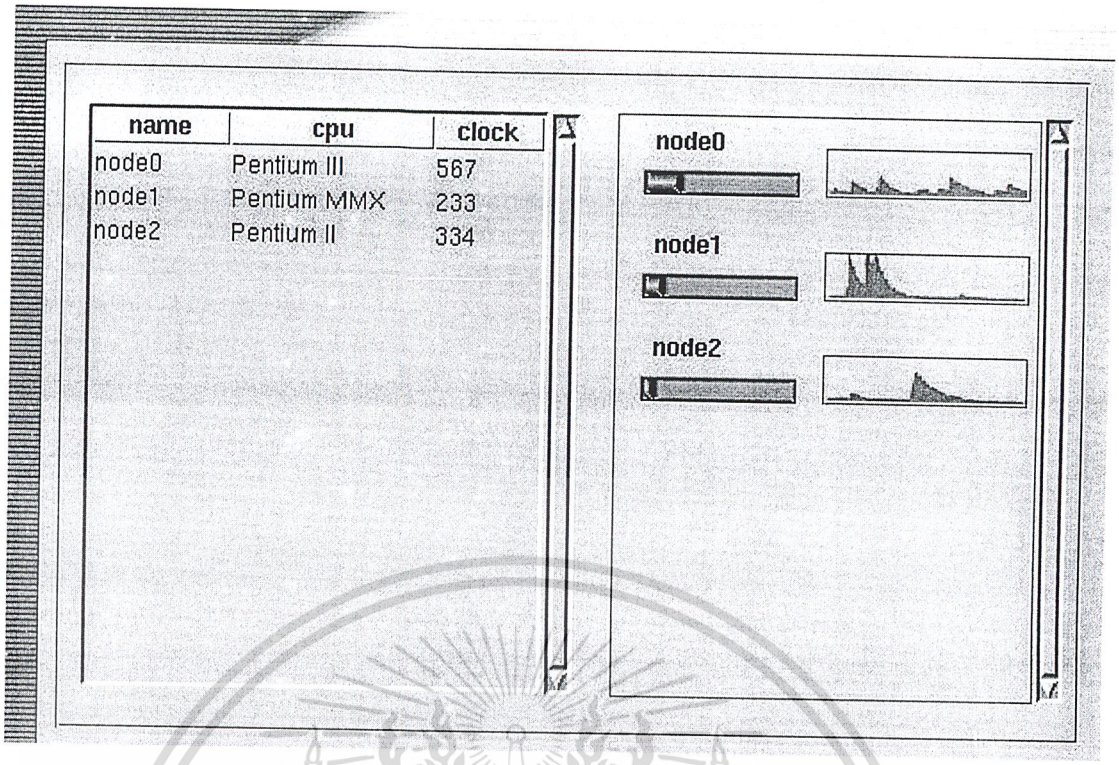
เมื่อออกแบบและเขียนโค้ดเสร็จแล้วเราจึงนำมาทดสอบกับระบบคลัสเตอร์ที่เราได้สร้างขึ้นมาก่อนหน้านี้และได้นำโปรแกรมตัวอย่าง 2-3 โปรแกรมมารันและทดสอบร่วมกันทั้งหมดได้ผลดังนี้



รูปที่ 7-1 โปรแกรมเอ็กซ์มอนิเตอร์



รูปที่ 7-2 การลบ,เพิ่ม หรือปรับแต่งโหนด สามารถทำได้ที่เมนู config

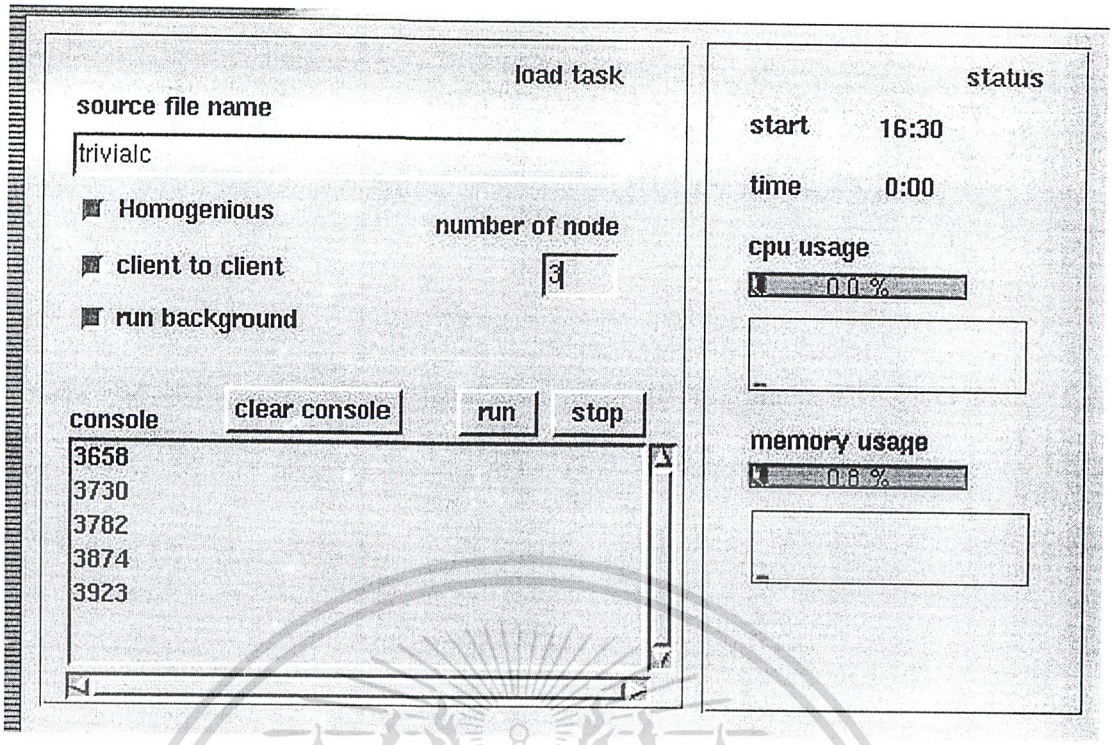


รูปที่ 7-3 ข้อมูลซีพียู, การเก็บบันทึกข้อมูลและ การพล้อตกราฟ

Memory Info

name	Total	Free	Shared	Buffer	Cached	SwapTotal	SwapFree
node0	63916	3192	31428	2164	18604	104380	100364
node1	63072	1824	44556	5472	22148	152608	150444
node2	62904	13080	18564	8860	28144	144576	141916

รูปที่ 7-4 ข้อมูลในส่วนของหน่วยความจำ



รูปที่ 7-5 การสั่งรันงาน และการมอนิเตอร์โปรแกรมเอ็มพีไอ

บทที่ 8
บทสรุปโปรแกรมมอนิเตอร์ระบบประมวลผลแบบกลุ่มลินุกซ์

8.1 ลักษณะของโปรแกรม

โครงงานนี้จัดทำขึ้นเพื่อศึกษาการทำงานของระบบลินุกซ์คลัสเตอร์และการประมวลแบบขนานบนลินุกซ์คลัสเตอร์ วิธีสร้างระบบลินุกซ์คลัสเตอร์ และวิธีการในการเฝ้าดูการทำงานของระบบ จากการทดลองและได้ประสบกับปัญหาหลายอย่าง จึงมีแนวคิดในการสร้างเครื่องมือที่จะมาช่วยการทำงานให้น้อยลงจนเกิดเป็นโปรแกรมมอนิเตอร์ระบบลินุกซ์คลัสเตอร์ขึ้นมา เมื่อนำมาใช้งานจริงพบว่าการทำงานหลายๆอย่างง่ายขึ้น เห็นภาพพจน์ชัดเจนขึ้น เนื่องจากมีการเก็บข้อมูลมาพล้อตเป็นกราฟ รวมทั้งโปรแกรมมีการจัดการหลายอย่างให้อัตโนมัติ ทำให้การทำงานประหยัดเวลาขึ้น การติดตั้งโปรแกรมไม่ยุ่งยากและไม่ต้องคอมไพล์ ใดๆทั้งสิ้น แต่ก็มีข้อจำกัดคือไม่สามารถปรับอัตราการอัพเดทข้อมูลให้สูงได้ เนื่องจากการใช้รีโหมตเชลล์ต้องมีช่วงเวลาในการเก็บข้อมูลของแต่ละโหนดระยะหนึ่ง หากมีจำนวนโหนดมากก็จะทำให้การเก็บข้อมูลทั้งหมดช้าลงไปด้วย อีกทั้งการเก็บข้อมูลบ่อยๆทำให้สิ้นเปลืองแบนด์วิทของการสื่อสารระหว่างโหนดอีกด้วย

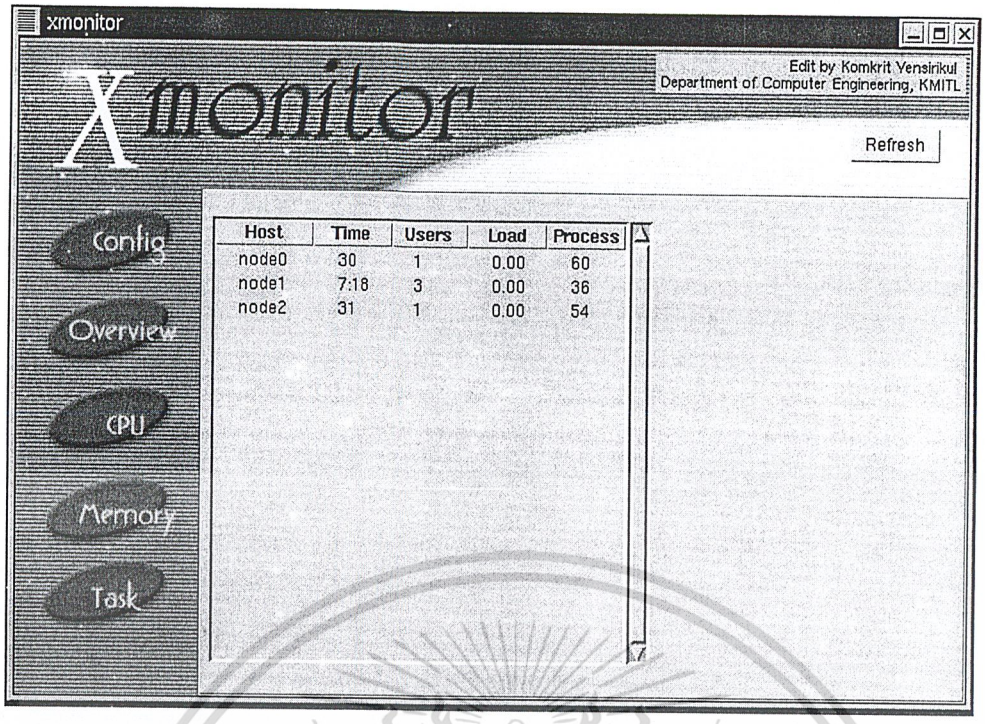
8.2 การใช้งาน

8.2.1 เริ่มต้นใช้งานโปรแกรม

เข้าระบบปฏิบัติการลินุกซ์ เปิดเอ็กซ์วินโดว์ขึ้นมา เข้าไปที่เทอร์มินอล(terminal) พิมพ์คำสั่งที่ไดเร็กทอรี่ที่เราติดตั้งโปรแกรมไว้ดังต่อไปนี้

```
./xmonitor.tcl
```

โปรแกรมจะเริ่มทำงานโดยมีหน้าตาดังนี้



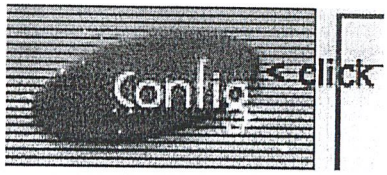
รูปที่ 8-1 หน้าตาของโปรแกรมตอนเริ่มต้นทำงาน

จากรูปที่ 8.1 จะเป็นหน้าตาตอนเริ่มต้นการทำงานของโปแแกรมมอนิเตอร์ระบบประมวลผลแบบกลุ่มลินุกซ์ โดยทางซ้ายมือจะมีเมนูให้เลือก 5 เมนูด้วยกันซึ่งประกอบด้วย Config, Overview, CPU, Memory และ Task และตอนเริ่มต้นโปรแกรมค่าดีฟอลต์คือจะเข้าไปที่เมนูของ Overview ซึ่งจะทำงานดูสภาพความพร้อมและข้อมูลของระบบโดยทั่วๆไปออกมา ถ้าโหนดใดไม่ทำงานก็จะไม่แสดงค่าเป็นตัวเลขออกมา ทำให้เรารู้ว่าโหนดนั้นไม่พร้อมที่จะทำงาน ค่าต่างๆที่น่าสนใจมีความหมายดังนี้

- Host ชื่อโหนด
- Time เวลาที่โหนดเริ่มทำงาน
- Users จำนวนผู้ใช้งานในแต่ละโหนด
- Load ภาระของซีพียูเทียบเป็นเปอร์เซ็นต์ของ cpu time
- Process จำนวนโปรเซสในแต่ละโหนด

8.2.2 ปรับแต่งโหนดในคลัสเตอร์

เข้าเมนู config โดยกดปุ่ม config ดังรูป

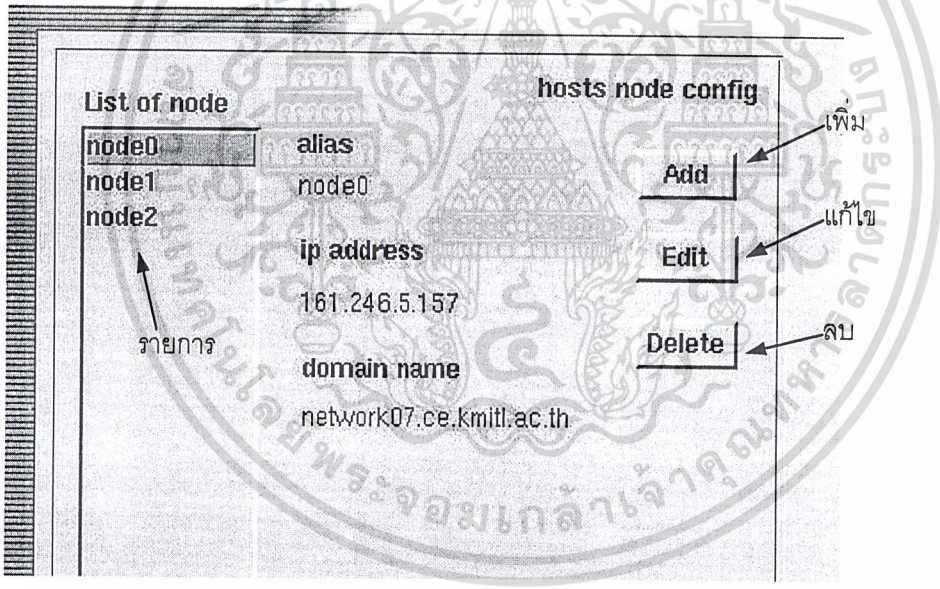


รูปที่ 8-2 เมนู config

เมื่อเข้าเมนู config แล้ว กดปุ่ม add ใส่ค่าต่างๆดังนี้

- alias ชื่อโหนด
- ip address ใส่ไอพีแอดเดรสของโหนด
- domain name ใส่ชื่อเต็มรวมโดเมนเนมของโหนด

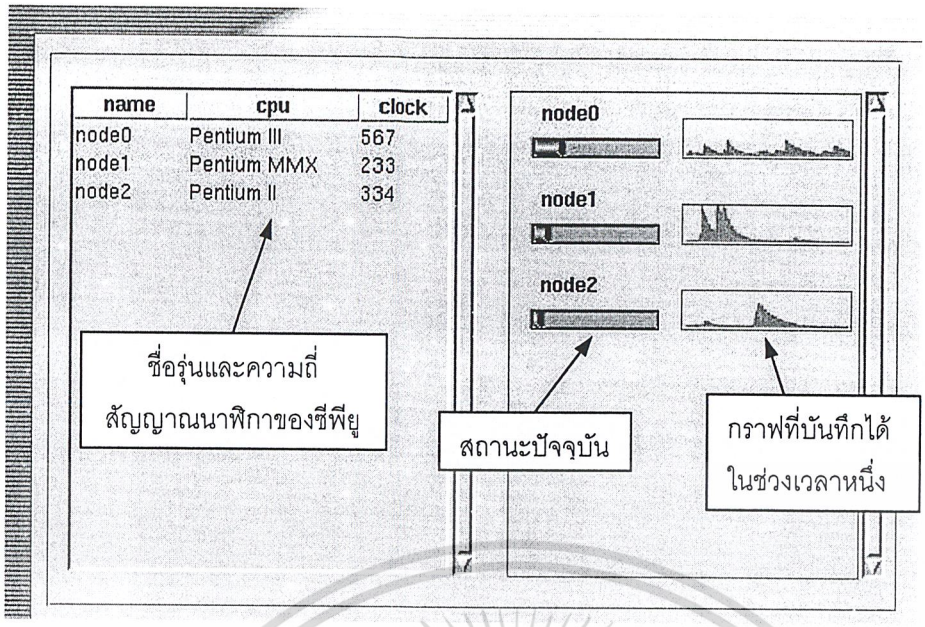
เสร็จแล้วทำการกด ok เพื่อยืนยันการบันทึกข้อมูล นอกจากนั้นเรายังสามารถลบโหนดใดๆได้ โดยการเลือกโหนดในรายการทางซ้าย แล้วกด delete หรือทำการแก้ไขข้อมูลของโหนดโดยการกด edit ดังรูป



รูปที่ 8-3 การเพิ่ม, ลบ และแก้ไขโหนด

8.2.3 ข้อมูลของซีพียู

เราสามารถดูข้อมูลซีพียูได้โดยการคลิ้กไปที่เมนู CPU ส่วนนี้จะทำหน้าเก็บบันทึกและแสดงข้อมูลการใช้ซีพียูออกมาเป็นกราฟ โดยคิดจากเปอร์เซ็นต์การทำงานของซีพียูเทียบกับ cpu time ทั้งหมด ดังรูป



รูปที่ 8-4 ข้อมูลซีพียู

8.2.4 ข้อมูลของหน่วยความจำ

คลิ๊กที่เมนู memory จะเข้าสู่ส่วนของหน่วยความจำ ซึ่งเราสามารถดูรายละเอียดของหน่วยความจำแต่ละโหนดได้ดังรูป

Memory Info

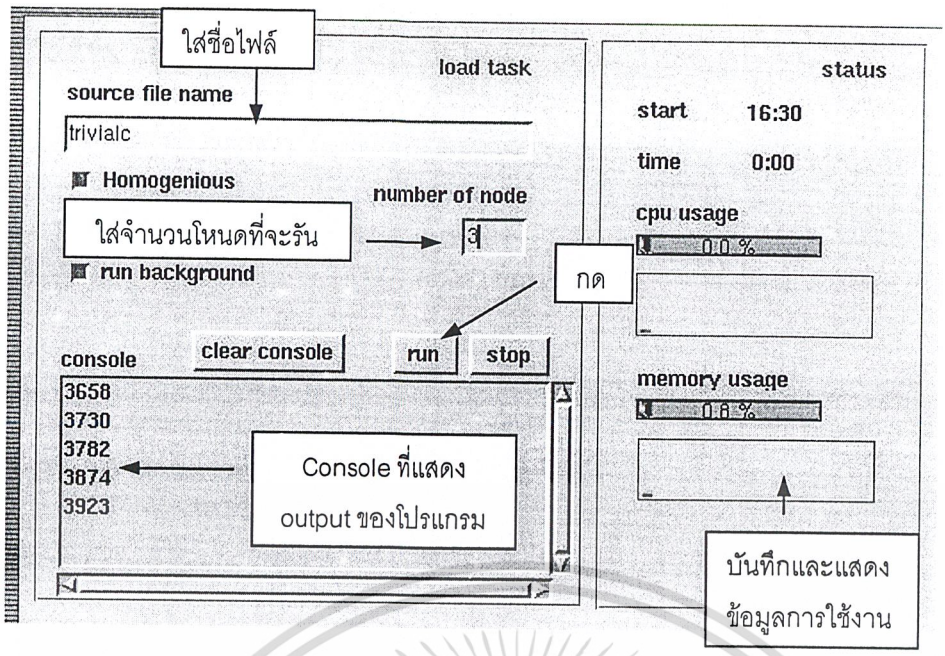
name	Total	Free	Shared	Buffer	Cached	SwapTotal	SwapFree
node0	63916	3192	31428	2164	18604	104380	100364
node1	63072	1824	44556	5472	22148	152608	150444
node2	62904	13080	18564	8860	28144	144576	141916

รูปที่ 8-5 ข้อมูลหน่วยความจำ

8.2.5 การจัดการงาน

คลิ๊กที่เมนู task เพื่อเข้าสู่ส่วนของการจัดการงาน ที่เป็นโปรแกรมเชิงขนานที่เขียนโดยใช้เอ็มพีไอไลบรารี่ การสั่งรันงานทำโดยใส่ชื่อไฟล์ในช่องของ source file name, ใส่จำนวนโหนดในช่องของ number of node แล้วกด run

งานบางอย่างอาจมีเอาท์พุตออกมาด้วย ซึ่งจะปรากฏในส่วนของ console โดยที่เมื่องานยังไม่เสร็จก็จะทำการบันทึกการใช้ซีพียูกับหน่วยความจำไว้ด้วยดังรูป



รูปที่ 8-6 การจัดการงาน

8.3 สรุปผลและแนวทางในการพัฒนาต่อ

โครงงานนี้ได้สำเร็จตามเป้าหมายในระดับหนึ่ง เครื่องมือที่สร้างขึ้นสามารถใช้งานได้ดีพอสมควร ความง่ายในการใช้งานและติดตั้งยังเป็นเป้าหมายหลักของโครงงานนี้ แนวทางในการพัฒนาโปรแกรมมอนิเตอร์ระบบลินุกซ์คลัสเตอร์ให้ดีขึ้นมีดังนี้

8.3.1 พัฒนาไปเป็นเครื่องมือสำหรับผู้ดูแลระบบ (Administration tool)

โดยเพิ่มความสามารถในการติดตั้ง (installation) และจัดการระบบคลัสเตอร์เช่น เพิ่มโหนด, ลบโหนด, ปรับแต่งโหนด โดยทำเป็น package ที่สามารถติดตั้งได้ในขั้นตอนเดียว (one time installation) ไม่ต้องมีการติดตั้งไลบรารี่อื่นๆเพิ่มเติม ซึ่งจะช่วยลดขั้นตอนของการดูแลระบบลงได้มาก

8.3.2 เพิ่มความสามารถในการวัดประสิทธิภาพของระบบ

นอกจากสามารถดูข้อมูลทั่วๆไปของระบบแล้ว ข้อมูลที่จำเป็นอีกอย่างหนึ่งของการมอนิเตอร์คือประสิทธิภาพของระบบ (performance) ซึ่งสามารถวัดได้ด้วยโปรแกรมตรวจวัด (benchmark) ทั่วๆไป หากนำข้อมูลเหล่านี้แสดงร่วมกับข้อมูลเดิมที่ได้จากโปรแกรมมอนิเตอร์แล้ว ก็จะเป็นประโยชน์กับผู้ใช้งานมากขึ้น

ภาคผนวก ก

SCRIPT SDCT

```
#!/bin/bash
#-----------------------------------------------------------------------
# Set-up Disk-less Client Template
#-----------------------------------------------------------------------
# $Id: sdct,v 1.11 1999/04/29 12:01:53 root Exp $
# /usr/local/sbin/sdct
# ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils/disk-less
# Originally known as 'setup_template'
#
# Version     : 0.0.5
#
# Date        : 24 April 1999
#
# Author      : Jacek Radajewski
#             jacek@usq.edu.au
#
# OS          : Red Hat Linux 5.2
#
# Purpose           : This script is used to setup an NFS-root  template Beowulf
#                     disk-less client.  This Template will be used to create the
#                     NFS root directory for each of the disk-less clients.
#             assumes Red Hat Linux 5.2, and that the /tftpboot
#             directory is on the same file system as /etc and all
#             of its sub directories.
#-----------------------------------------------------------------------

show_help () {

   echo -e "sdct options\n"
   echo -e "[-t tftpboot_dir] : Specify alternative tftpboot directory"
```

```
    echo -e "              If this option is ommited, /tftpboot will be used"
    echo -e "[-s server]     : Name or the IP address of the NFS-root server."
    echo -e "              If this option is ommited /bin/hostname will be used to"
    echo -e "              obtain the host name of the server."
    echo -e "              NOTE : The name of the interface to the cluster might not"
    echo -e "              be the same as the name returned by the hostname program."
    echo -e "[-h]          : This help message"
    exit 0
}

# before we do anything we check if we are the superuser

if [ $UID != '0' ] ; then
    echo "Only root can create disk-less client template."
    echo "Giving up."
    exit 1
fi

# set few default values before processing command line arguments

export SERVER=$(/bin/hostname)
export TFTPBOOT=/tftpboot

# process command line options

while getopts ":fhs:t:" opt; do
    case $opt in
        t ) export TFTPBOOT=$OPTARG;;
        s ) export SERVER=$OPTARG;;
        h ) show_help;;
        \? ) echo 'adcn -h for help'
            exit 1
    esac
done
```

```
export TEMPLATE=$TFTPBOOT/Template

if [ -d "$TFTPBOOT" ] ; then
   echo "Directory $TFTPBOOT exists.  Please remove it and try again."
   exit 1
fi

# we first create all the directories

/bin/mkdir $TFTPBOOT
/bin/mkdir $TEMPLATE
/bin/mkdir $TEMPLATE/bin
/bin/mkdir $TEMPLATE/boot
/bin/mkdir $TEMPLATE/dev
/bin/mkdir $TEMPLATE/etc
/bin/mkdir $TEMPLATE/home
/bin/mkdir $TEMPLATE/lib
/bin/mkdir $TEMPLATE/mnt
/bin/mkdir $TEMPLATE/root
/bin/mkdir $TEMPLATE/sbin
/bin/mkdir $TEMPLATE/tmp
/bin/mkdir $TEMPLATE/var

/bin/ln -sf /usr $TEMPLATE/usr
/bin/ln -sf /proc $TEMPLATE/proc

# now we copy all the files

echo "Creating NFS-Root template directory structure ... "

echo -n "/bin "
cd /bin
```

```
/bin/tar -cpf - * | (cd $TEMPLATE/bin ; tar -xpf -)

echo -n "/boot "
cd /boot
/bin/tar -cpf - * | (cd $TEMPLATE/boot ; tar -xpf -)

echo -n "/dev "
cd /dev
/bin/tar -cpf - * | (cd $TEMPLATE/dev ; tar -xpf -)

echo -n "/etc "
cd /etc
/bin/tar -cpf - * | (cd $TEMPLATE/etc ; tar -xpf -)

echo -n "/lib "
cd /lib
/bin/tar -cpf - * | (cd $TEMPLATE/lib ; tar -xpf -)

echo -n "/mnt "
cd /mnt
/bin/tar -cpf - * | (cd $TEMPLATE/mnt ; tar -xpf -)

echo -n "/root "
cd /root
/bin/tar -cpf - * | (cd $TEMPLATE/root ; tar -xpf -)

echo -n "/sbin "
cd /sbin
/bin/tar -cpf - * | (cd $TEMPLATE/sbin ; tar -xpf -)

echo -n "/var "
cd /var
/bin/tar -cpf - * | (cd $TEMPLATE/var ; tar -xpf -)
```

```
echo -e -n "\nCleaning up ..."

# remove network scripts

/bin/rm -f $TEMPLATE/etc/sysconfig/network
/bin/rm -f $TEMPLATE/etc/sysconfig/network-scripts/ifcfg-eth*

# remove fstab and mtab

/bin/rm -f $TEMPLATE/etc/fstab
/bin/rm -f $TEMPLATE/etc/mtab

#-----------------------------------------------------------------
# remove old and create new, empty log files.
# most of these will not be used because most messages
# will be logged remotely to the server via syslogd
#-----------------------------------------------------------------

/bin/rm -f $TEMPLATE/var/log/secure*
/bin/touch $TEMPLATE/var/log/secure

/bin/rm -f $TEMPLATE/var/log/messages*
/bin/touch $TEMPLATE/var/log/messages

/bin/rm -f $TEMPLATE/var/log/maillog*
/bin/touch $TEMPLATE/var/log/maillog

/bin/rm -f $TEMPLATE/var/log/cron*
/bin/touch $TEMPLATE/var/log/cron

/bin/rm -f $TEMPLATE/var/log/lastlog*
/bin/touch $TEMPLATE/var/log/lastlog

/bin/rm -f $TEMPLATE/var/log/xferlog*
```

```
/bin/touch $TEMPLATE/var/log/xferlog

/bin/rm -f $TEMPLATE/var/log/dmesg*
/bin/touch $TEMPLATE/var/log/dmesg

# clients should not run samba

/bin/rm -f $TEMPLATE/var/log/log.nmb
/bin/rm -f $TEMPLATE/var/log/log.smb
/bin/rm -rf $TEMPLATE/var/log/samba*

# clients should not run any cron jobs

/bin/rm -f $TEMPLATE/etc/cron.hourly/*
/bin/rm -f $TEMPLATE/etc/cron.daily/*
/bin/rm -f $TEMPLATE/etc/cron.weekly/*
/bin/rm -f $TEMPLATE/etc/cron.monthly/*

/bin/rm -f $TEMPLATE/var/spool/cron/*

#-----------------------------------------------------------------
# remove some of the services from /etc/rc.d/rc3.d
# you might want to change/add/remove lines here
#-----------------------------------------------------------------

/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*arpwatch
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*dhcpd
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*gated
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*gpm
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*httpd
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*innd
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*named
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*postgresql
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*gated
```

```
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*rarp
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*routed
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*sendmail
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*smb
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*snmpd
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*nfs
/bin/rm -f $TEMPLATE/etc/rc.d/rc3.d/S*portmap

#-----------------------------------------------------------------------
# The following is a bad hack, but it allows us to reboot disk-less clients.
# Reboot without unmounting NFS file systems, and without stopping the network
#-----------------------------------------------------------------------

/bin/rm -f $TEMPLATE/etc/rc.d/rc6.d/K*nfsfs
/bin/rm -f $TEMPLATE/etc/rc.d/rc6.d/K*network

# same for init 0

/bin/rm -f $TEMPLATE/etc/rc.d/rc0.d/K*nfsfs
/bin/rm -f $TEMPLATE/etc/rc.d/rc0.d/K*network

#-----------------------------------------------------------------------
# create a new /etc/syslog.conf file
# we want to log everything on the server
# you will HAVE TO run syslogd on the server with "-r"
# to enable clients to log their messages on the server
# this script doesn't do it.  Modify /etc/rc.d/init.d/syslog :
#-----------------------------------------------------------
#case "$1" in
#  start)
#       echo -n "Starting system loggers: "
#       daemon syslogd -r

/bin/rm -f $TEMPLATE/etc/syslog.conf
```

```
echo "*.* @$SERVER" >> $TEMPLATE/etc/syslog.conf


#-----------------------------------------------------------------------
# there are files which we want to be the same on the server and all clients
# /etc/passwd for example has to be the same on all machines.
# we want our $TEMPLATE/etc/passwd to be a hard link to /etc/passwd
#-----------------------------------------------------------------------

/bin/ln -f /etc/passwd      $TEMPLATE/etc/passwd
/bin/ln -f /etc/group       $TEMPLATE/etc/group
/bin/ln -f /etc/issue       $TEMPLATE/etc/issue
/bin/ln -f /etc/issue.net   $TEMPLATE/etc/issue.net
/bin/ln -f /etc/profile     $TEMPLATE/etc/profile
/bin/ln -f /etc/bashrc      $TEMPLATE/etc/bashrc
/bin/ln -f /etc/hosts       $TEMPLATE/etc/hosts
/bin/ln -f /etc/hosts.equiv $TEMPLATE/etc/hosts.equiv

#-----------------------------------------------------------------------
# create rc script to setup rarp entries
# this script shoud run in both runlevel 3 and 5
#-----------------------------------------------------------------------

/bin/rm -f /etc/rc.d/init.d/rarp
/bin/touch /etc/rc.d/init.d/rarp
/bin/chmod u+x /etc/rc.d/init.d/rarp
/bin/ln -sf /etc/rc.d/init.d/rarp /etc/rc.d/rc3.d/S95rarp
/bin/ln -sf /etc/rc.d/init.d/rarp /etc/rc.d/rc5.d/S95rarp

# remove linuxconf from /bin in the template

rm -f $TEMPLATE/bin/linuxconf

echo " done."
```

```
echo -e "\nYou can now use the adcn script to create a file system for each of the clients"
echo "For help on adcn type: adcn -h"
```

ภาคผนวก ข

SCRIPT ADCN

```
#!/bin/bash
#-------------------------------------------------------------------------
# Add Disk-less Client Node
#-------------------------------------------------------------------------
# $Id: adcn,v 1.16 1999/05/08 11:39:55 jacek Exp jacek $
# /usr/local/sbin/adcn
# ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils/disk-less
# Originally know as 'add_node'
#
# Version   : 0.0.8
#
# Date      : 27 April 1999
#
# Authors   : Jacek Radajewski <jacek@usq.edu.au>
#           Tony Nugent <nugent@usq.edu.au>
#
# Copyright : (C) Jacek Radajewski 1997-1999
#           This program is distributed under GNU GENERAL PUBLIC LICENSE
#           Version 2, June 1991 Copies of the license can be down found at
#           http://www.fsf.org/copyleft/gpl.html
#
# Thanx to  : Tony Nugent for many great ideas
#
#
# Purpose   : This script adds a new node to a disk-less-client Beowulf cluster
#           adcn assumes Red Hat 5.2 and might not work with other versions
#           of Red Hat or other distributions of Linux . adcn uses the
#           /tftpboot/Template directory to create NFS root file system
#           for the client node. /tftpboot/Template should be created with
#           the sdct script (originally known as setup_template), which can
```

```
#          be found at ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils/
#
# Command line options : check the show_help () function
#
#-----------------------------------------------------------------------


#-----------------------------------------------------------------------
# show help on usage
#-----------------------------------------------------------------------

show_help () {

cat << EOF

adcn options

[-f]          : force installation even if adcn does not support this
                distribution of Linux or version of Red Hat Linux

-i ip_address   : IP Addrees of the disk-less client node"

[-m mac_address] : MAC or hardware address of the disk-less client node

-l            : Listen for RARP request on the interface specified with -D
                and use the sniffed MAC address as client's harware address.
                If -m is specified with this option, given MAC address is
                ignored. (not implemented)

-c client_name  : Host name of the disk-less client node.  This must the first
                name of the node and not the fully qualified domain name

[-d domain_name] : Domain name.  If not specified servers domain will be used.
```

[-n netmask] : Netmask address.

-s server : Default gateway and the NFS server for the disk-less client node.
specified, server node is used as the default gateway

[-g gateway] : default gateway address for the client. If not specified server's address will be used

[-N net_address] : Network address

[-b broadcast] : Broadcast address

[-D interface] : Use the interface configuration to figure out client's netmask, broadcast, and gateway addresses. This is also used by the -l option to tell tcpdump on which interface to listen for the RARP request.

[-h] : This help message

Example :

adcn -i 10.0.0.1 -c node1 -d beowulf.my.domain -D eth1 -l

EOF
exit 0
}

set few default values before processing command line arguments

FORCE=false
LISTEN=false

```
DOMAIN=$(/bin/dnsdomainname)
HOST=none
CLIENT_IP=none
MACADDR=none
INTERFACE=none
DEVICE=none
NETMASK=none
BROADCAST=none
NETWORK=none
SERVER_IP=none
CLIENT_GW=none
PATH=/bin:/sbin:/usr/bin:/usr/sbin

#------------------------------------------------------------------
# process command line options
#------------------------------------------------------------------

while getopts ":fhld:i:m:c:n:s:N:b:g:D:" opt; do
   case $opt in
          f ) FORCE=true;;
          d ) DOMAIN=$OPTARG;;
          i ) CLIENT_IP=$OPTARG;;
          m ) MACADDR=$OPTARG;;
          c ) HOST=$OPTARG;;
          n ) NETMASK=$OPTARG;;
          s ) SERVER_IP=$OPTARG;;
          N ) NETWORK=$OPTARG;;
          b ) BROADCAST=$OPTARG;;
          g ) CLIENT_GW=$OPTARG;;
          l ) LISTEN=true;;
          D ) INTERFACE=$OPTARG;;
          h ) show_help;;
          \? ) echo 'adcn -h for help'
              exit 1
```

```
    esac
  done

  #---------------------------------------------------------------------
  # before we do anything we check if we are the superuser
  #---------------------------------------------------------------------

  if [ $UID != '0' ] ; then
    echo "Only root can add disk-less clients."
    echo "Giving up."
    exit 1
  fi

  #---------------------------------------------------------------------
  # we check if interface was specified, if so we source the configuration file
  #---------------------------------------------------------------------

  if [ $INTERFACE != 'none' ] ; then
    if [ -f /etc/sysconfig/network-scripts/ifcfg-$INTERFACE ] ; then
      . /etc/sysconfig/network-scripts/ifcfg-$INTERFACE
    else
        echo "Cannot find /etc/sysconfig/network-scripts/ifcfg-$INTERFACE"
        echo "Giving Up."
        exit 1
    fi
  fi


  #---------------------------------------------------------------------
  # Quick check to make sure that the device specified in the configuration
  # file is consistent with the file name; are we looking at the correct dev ?
  #---------------------------------------------------------------------

  if [ $INTERFACE != $DEVICE ] ; then
```

```
    echo "Possible error in /etc/sysconfig/network-scripts/ifcfg-$INTERFACE"
    echo "Giving up."
    exit 1
  fi

  #---------------------------------------------------------------------
  # check if we need to get the MAC address our self
  # we use tcpdump to get sniff the MAC address
  # this is a hack; suggestions welcome :)
  #---------------------------------------------------------------------

  if [ $LISTEN != 'false' ] ; then
    if [ -f /usr/sbin/tcpdump ] ; then
        echo -n "Listening for clients RARP request ..."
        rm -f /tmp/rarp.sniff
        tcpdump rarp -c 1 -i $INTERFACE > /tmp/rarp.sniff 2> /dev/null
        read a b c mac d e < /tmp/rarp.sniff
        echo " got $mac"
        export MACADDR=$mac
    else
        echo "Can't find tcpdump, but need it for the -l option :("
        echo "Giving up."
        exit 1
    fi
  fi

  #---------------------------------------------------------------------
  # check if everything is set
  #---------------------------------------------------------------------

  if [ $HOST = 'none' ] || [ $CLIENT_IP = 'none' ] || [ $MACADDR = 'none' ] ; then
    echo '1: adcn -h for help'
    exit 1
  fi
```

```
if [ $CLIENT_GW = 'none' ] || [ $SERVER_IP = 'none' ] ; then
   echo '2: adcn -h for help'
   exit 1
fi

#-------------------------------------------------------------------------
# make sure that the host name is not a FQDN
#-------------------------------------------------------------------------

echo $HOST | grep "\." > /dev/null 2>&1
if [ $? = 0 ] ; then
   echo "Host name cannot be a fully qualified domain name.  Use -d to set domain."
   exit 1
fi

#-------------------------------------------------------------------
# simple and incomplete check of the MAC address
#-------------------------------------------------------------------

if [ $(echo $MACADDR | cut -f 6 -d ":")test = "test" ] ; then
   echo "$MACADDR is not a valid MAC address"
   exit 1
fi

echo "Adding disk-less client $HOST to the cluster ... "

#-------------------------------------------------------------------------
# check linux release but not if forced install
# at this stage only check for Red Hat 5.2
#-------------------------------------------------------------------------

if [ $FORCE = 'false' ] ; then
   if [ -f /etc/redhat-release ] ; then
```

```
        cat /etc/redhat-release | grep "Apollo" > /dev/null 2>&1
        if [ $? = 0 ] ; then
           export REDHAT=Apollo
        else
           echo "This script might not work with $(cat /etc/redhat-release)"
           echo "Use -f to force installation"
           exit 1
        fi
    else
        echo "Can't find /etc/redhat-release file."
        echo "Not a Red Hat Linux distribution.  Giving up"
        echo "Use -f to force installation.  Good luck :)"
        exit 1
    fi
fi

#---------------------------------------------------------------------
# the location of our template
# you must create template before you run this script !!!
# use the sdct script to create the template
# and then customize it for your needs
#---------------------------------------------------------------------

TEMPLATE_DIR=/tftpboot/Template

DEST_DIR=/tftpboot/$CLIENT_IP

#---------------------------------------------------------------------
# check if this IP address already exists
#---------------------------------------------------------------------

if [ -f $DEST_DIR ] ; then
   echo "$DEST_DIR entry exists.  Giving up."
```

```
    exit 1
fi

# do few other checks before going ahead

if [ ! -d $TEMPLATE_DIR ] ; then
    echo "Cannot find Template directory $TEMPLATE_DIR.  Giving up."
    exit 1
fi


# create the root directory for new node

/bin/mkdir /tftpboot/$CLIENT_IP

# we also want the name of the new node to be a link to the real (IP address)
# directory

/bin/ln -sf /tftpboot/$CLIENT_IP /tftpboot/$HOST
/bin/ln -sf /tftpboot/$CLIENT_IP /tftpboot/$HOST.$DOMAIN

# create sub directories

/bin/mkdir $DEST_DIR/bin
/bin/mkdir $DEST_DIR/boot
/bin/mkdir $DEST_DIR/dev
/bin/mkdir $DEST_DIR/etc
/bin/mkdir $DEST_DIR/home
/bin/mkdir $DEST_DIR/lib
/bin/mkdir $DEST_DIR/mnt
/bin/mkdir $DEST_DIR/proc
/bin/mkdir $DEST_DIR/root
/bin/mkdir $DEST_DIR/sbin
/bin/mkdir $DEST_DIR/tmp
```

```
/bin/mkdir $DEST_DIR/usr
/bin/mkdir $DEST_DIR/var

/bin/chmod 777 $DEST_DIR/tmp

echo "Building NFS root file system ..."

#-----------------------------------------------------------------------
# we now create the files in all of the directories
# we make hard links to files in our Template directory
#-----------------------------------------------------------------------

#-----------------------------------------------------------------------
#                       /bin
#-----------------------------------------------------------------------

echo -n " /bin "
cd $TEMPLATE_DIR/bin
for directory in $(/usr/bin/find . -not -name '.' -type d) ; do
   /bin/mkdir $DEST_DIR/bin/$directory
done
for file in $(/usr/bin/find . -type f -maxdepth 1 -follow) ; do
   /bin/ln $TEMPLATE_DIR/bin/$file $DEST_DIR/bin/$file
done

#-----------------------------------------------------------------------
#                       /boot
#-----------------------------------------------------------------------

echo -n " /boot "
cd $TEMPLATE_DIR/boot
for directory in $(/usr/bin/find . -not -name '.' -type d) ; do
   /bin/mkdir $DEST_DIR/boot/$directory
done
```

```
for file in $(/usr/bin/find . -type f -maxdepth 1) ; do
   /bin/ln $TEMPLATE_DIR/boot/$file $DEST_DIR/boot/$file
done

#-----------------------------------------------------------------------
#                          /dev
#-----------------------------------------------------------------------

echo -n " /dev "
cd $TEMPLATE_DIR/dev

# Linking of devices is still very shaky.  There are problems
# with linking /dev/fd, /dev/stdin, /dev/stderr, and /dev/stdout
# directories.  These are done explicitly.

/bin/ln -sf ../proc/self/fd       $DEST_DIR/dev/fd
/bin/ln -f  $TEMPLATE_DIR/dev/stdin  $DEST_DIR/dev/stdin
/bin/ln -f  $TEMPLATE_DIR/dev/stdout $DEST_DIR/dev/stdout
/bin/ln -f  $TEMPLATE_DIR/dev/stderr $DEST_DIR/dev/stderr

for directory in $(/usr/bin/find . -not -name '.' -type d) ; do
   /bin/mkdir $DEST_DIR/dev/$directory
done
# link files  block, character, then normal files (e.g. MAKEDEV)
for file in $(/usr/bin/find . -type c -follow -mount) ; do
   /bin/ln -fd $TEMPLATE_DIR/dev/$file $DEST_DIR/dev/$file
done
for file in $(/usr/bin/find . -type b -follow -mount) ; do
   /bin/ln -fd $TEMPLATE_DIR/dev/$file $DEST_DIR/dev/$file
done
for file in $(/usr/bin/find . -type f -follow -mount) ; do
   /bin/ln -fd $TEMPLATE_DIR/dev/$file $DEST_DIR/dev/$file 2>/dev/null
done
```

```
#-----------------------------------------------------------------------
#                          /etc
#-----------------------------------------------------------------------

echo -n " /etc "
cd $TEMPLATE_DIR/etc
for directory in $(/usr/bin/find . -not -name '.' -type d) ; do
   /bin/mkdir $DEST_DIR/etc/$directory
done

# this will produce a lot of errors due to simlinks under /etc/httpd
# we'll create these links explicitly if apache was installed

for file in $(/usr/bin/find . -type f -follow) ; do
   /bin/ln -f $TEMPLATE_DIR/etc/$file $DEST_DIR/etc/$file 2>/dev/null
done

if [ -d /etc/httpd ] ; then
   /bin/ln -sf ../../usr/lib/apache $DEST_DIR/etc/httpd/modules
   /bin/ln -sf ../../var/logs/httpd $DEST_DIR/etc/httpd/logs
fi

#-----------------------------------------------------------------------
#                          /lib
#-----------------------------------------------------------------------

echo -n " /lib "
cd $TEMPLATE_DIR/lib
for file in $(/usr/bin/find . -type f -follow -maxdepth 1) ; do
   /bin/ln -f $TEMPLATE_DIR/lib/$file $DEST_DIR/lib/$file
done

#-----------------------------------------------------------------------
#                          /root
```

```
#-----------------------------------------------------------------------

echo -n " /root "
cd $TEMPLATE_DIR/root
for file in $(/usr/bin/find . -type f -maxdepth 1) ; do
   /bin/ln -f $TEMPLATE_DIR/root/$file $DEST_DIR/root/$file
done

#-----------------------------------------------------------------------
#                          /sbin
#-----------------------------------------------------------------------
echo -n " /sbin "
cd $TEMPLATE_DIR/sbin
for directory in $(/usr/bin/find . -not -name '.' -type d -follow) ; do
   /bin/mkdir $DEST_DIR/sbin/$directory
done
for file in $(/usr/bin/find . -type f  -follow) ; do
   /bin/ln -f $TEMPLATE_DIR/sbin/$file $DEST_DIR/sbin/$file
done

#-----------------------------------------------------------------------
#                          /var
#-----------------------------------------------------------------------

echo -n " /var "
cd $TEMPLATE_DIR/var
for directory in $(/usr/bin/find . -not -name '.' -type d -follow) ; do
   /bin/mkdir $DEST_DIR/var/$directory
done
for file in $(/usr/bin/find . -type f -follow -mount) ; do
   /bin/ln $TEMPLATE_DIR/var/$file $DEST_DIR/var/$file
done

#-----------------------------------------------------------------------
```

```
# Now we create network configuration files for the node
# First we remove all existing configuration scripts
#--------------------------------------------------------------------

echo -e "\nCreating network scripts ..."

# clean up ...
/bin/rm -f $DEST_DIR/etc/sysconfig/network
/bin/rm -f $DEST_DIR/etc/sysconfig/network-scripts/ifcfg-eth*

# /etc/HOSTNAME
echo "$HOST.$DOMAIN" > $DEST_DIR/etc/HOSTNAME

#--------------------------------------------------------------------
# create client's /etc/sysconfig/network
#--------------------------------------------------------------------

cat << EOF >> $DEST_DIR/etc/sysconfig/network
NETWORKING=yes
HOSTNAME=$HOST.$DOMAIN
DOMAINNAME=$DOMAIN
GATEWAY=$CLIENT_GW
GATEWAYDEV=eth0
EOF

#--------------------------------------------------------------------
# create client's /etc/sysconfig/network-scripts/ifcfg-eth0
# if you have more than one interface per node you will have
# to add/change script
#--------------------------------------------------------------------

cat << EOF >> $DEST_DIR/etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
CLIENT_IP=$CLIENT_IP
```

```
NETMASK=$NETMASK
NETWORK=$NETWORK
BROADCAST=$BROADCAST
ONBOOT=yes
BOOTPROTO=none
EOF

#-----------------------------------------------------------------------
# add the RARP entry
# Our server will use this to anwser client's RARP request and tell it
# it's IP address
#-----------------------------------------------------------------------

/sbin/rarp -s $CLIENT_IP $MACADDR

echo "/sbin/rarp -s $CLIENT_IP $MACADDR" >> /etc/rc.d/init.d/rarp

#-----------------------------------------------------------------------
# add the new machine to /etc/exports
#-----------------------------------------------------------------------

cat << EOF  >> /etc/exports
/tftpboot/$CLIENT_IP/ $HOST (rw,no_all_squash,no_root_squash)
/usr            $HOST (rw,no_all_squash,no_root_squash)
/bin            $HOST (rw,no_all_squash,no_root_squash)
/sbin           $HOST (rw,no_all_squash,no_root_squash)
/lib            $HOST (rw,no_all_squash,no_root_squash)
/home           $HOST (rw,no_all_squash,no_root_squash)
EOF

#-----------------------------------------------------------------------
# add the new machine to /etc/hosts
#-----------------------------------------------------------------------
```

```
echo -e "$CLIENT_IP\t\t$HOST.$DOMAIN $HOST" >> /etc/hosts

#-----------------------------------------------------------------------
# restart the nfsd and mountd daemons on the server
# but do a simple check first
#-----------------------------------------------------------------------

# Red Hat specific
#/etc/rc.d/init.d/portmap restart
#/etc/rc.d/init/d/nfs status

# check if portmap running
rpcinfo -p 1> /dev/null 2>&1 ||
   echo "Warning : portmap not running"

# check if mountd running
rpcinfo -p 2> /dev/null | grep -q mountd ||
   echo "Warning : mountd not running"

# check if nfsd running
rpcinfo -p 2> /dev/null | grep -q nfs &&
   exportfs ||
   echo "Warning : nfsd not running"


#-----------------------------------------------------------------------
# create /etc/fstab entry
#-----------------------------------------------------------------------

/bin/rm -f $DEST_DIR/etc/fstab

echo -e "$SERVER_IP:/tftpboot/$CLIENT_IP\t/\tnfs\thard,intr,rw" >> $DEST_DIR/etc/fstab
echo -e "$SERVER_IP:/usr\t/usr\t\tnfs\tsoft,intr,rw"   >> $DEST_DIR/etc/fstab
echo -e "$SERVER_IP:/bin\t/bin\t\tnfs\tsoft,intr,rw"   >> $DEST_DIR/etc/fstab
```

```
echo -e "$SERVER_IP:/sbin\t/sbin\t\tnfs\tsoft,intr,rw"  >> $DEST_DIR/etc/fstab
echo -e "$SERVER_IP:/lib\t/lib\t\tnfs\tsoft,intr,rw"   >> $DEST_DIR/etc/fstab
echo -e "$SERVER_IP:/home\t/home\t\tnfs\thard,intr,rw"  >> $DEST_DIR/etc/fstab

/bin/rm -f $DEST_DIR/etc/mtab
/bin/touch $DEST_DIR/etc/mtab

# some other files have to be unique for each client


/bin/rm -f $DEST_DIR/var/log/wtmp*
/bin/touch $DEST_DIR/var/log/wtmp

/bin/rm -f $DEST_DIR/var/log/lastlog*
/bin/touch $DEST_DIR/var/log/lastlog



echo "Done."
```

ภาคผนวก ค

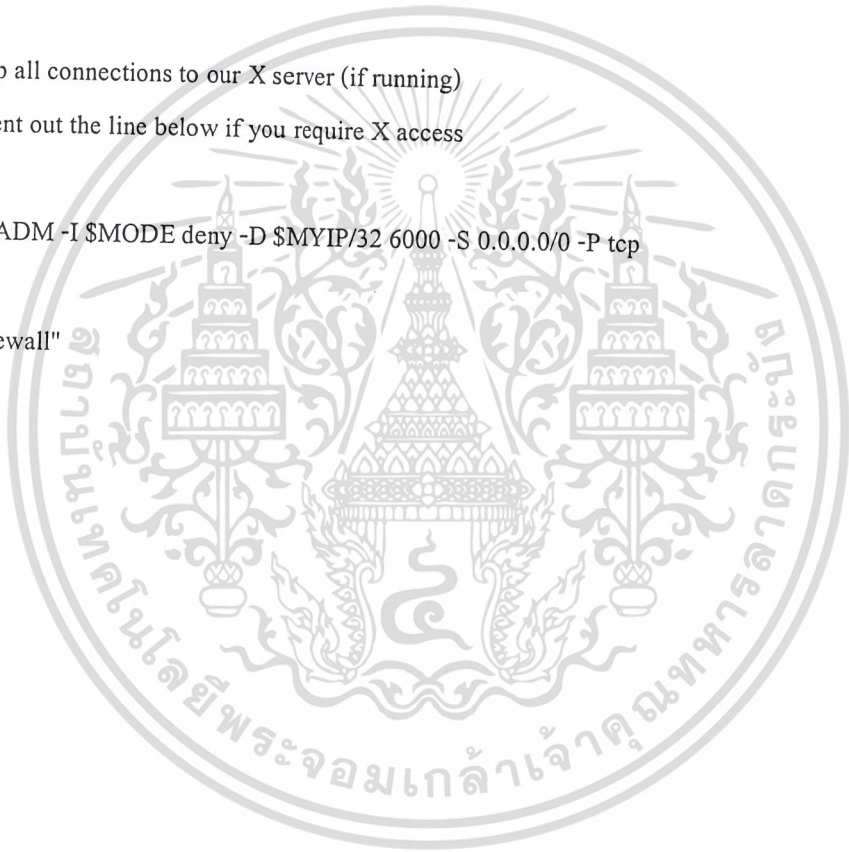
FIREWALL SCRIPT

```
# /etc/rc.d/init.d/firewall
#
# This file sets up the firewall rulz
# for topcat.eng.usq.edu.au Beowulf class supercomputer
# version 1.0.0
# 18/08/1998
#
# author : Jacek Radajewski jacek@usq.edu.au
#
# this is our third line of defence
# 1. most of the services are disabled in inted
# 2. secondly we use tcpd
# 3. we filter packets at the kernel level (this rc script)
#

# the ipfwadm program
IPFWADM="/sbin/ipfwadm"

case "$1" in
start)
echo -n "Inserting firewall rules ... "
export MODE="-i"
# default policies
export IN_POLICY="accept"
export OUT_POLICY="accept"
# if you have machines outside the cluster connected to
# the main system via IP tunnel as described at
# http://www.sci.usq.edu.ay/staff/jacek/topcat then you will
# have to allow forwarding
export FORWARD_POLICY="deny"
```

```
        ;;
 stop)
 echo -n "Deleting firewall rules ... "
 export MODE="-d"
 # default policies
 export IN_POLICY="accept"
 export OUT_POLICY="accept"
 export FORWARD_POLICY="accept"
        ;;
  *)
 echo "Usage: firewall {start|stop}"
 exit 1
 esac

 # source eth0 configuration
 # we assume that eth0 is our interface to the outside world
 # most firewall rules will be based on this

 . /etc/sysconfig/network-scripts/ifcfg-eth0

 # this must be set to the host's IP address
 export MYIP=$IPADDR

 # we want to allow administrator to telnet in
 export ADMINIP=139.x.x.x

 #--------------------------------------------------------------------
 # we first set default policies
 #--------------------------------------------------------------------

 $IPFWADM -I -p $IN_POLICY
 $IPFWADM -O -p $OUT_POLICY
 $IPFWADM -F -p $FORWARD_POLICY
```

```
#------------------------------------------------------------------
# forwarding rules
# deny all TCP and UDP
#------------------------------------------------------------------

$IPFWADM -F $MODE deny -S 0.0.0.0/0 -D 0.0.0.0/0 -P tcp
$IPFWADM -F $MODE deny -S 0.0.0.0/0 -D 0.0.0.0/0 -P udp

#------------------------------------------------------------------
# We go through the normal services and deny everything we don't need
# from outside.
#------------------------------------------------------------------

# ftp
#$IPFWADM -I $MODE deny -D $MYIP/32 ftp -S 0.0.0.0/0 -P tcp
#$IPFWADM -I $MODE accept -D $MYIP/32 ftp -S $ADMINIP/32 -P tcp

# telnet
#$IPFWADM -I $MODE deny  -D $MYIP/32 telnet -S 0.0.0.0/0 -P tcp
#$IPFWADM -I $MODE accept -D $MYIP/32 telnet -S $ADMINIP/32 -P tcp

# we block other known services ... well most of them

$IPFWADM -I $MODE deny -D $MYIP/32 echo -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 echo -S 0.0.0.0/0 -P udp
$IPFWADM -I $MODE deny -D $MYIP/32 discard -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 discard -S 0.0.0.0/0 -P udp
$IPFWADM -I $MODE deny -D $MYIP/32 systat -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 daytime -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 daytime -S 0.0.0.0/0 -P udp
$IPFWADM -I $MODE deny -D $MYIP/32 netstat -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 finger -S 0.0.0.0/0 -P tcp
#$IPFWADM -I $MODE deny -D $MYIP/32 http -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 pop -S 0.0.0.0/0 -P tcp
```

```
$IPFWADM -I $MODE deny -D $MYIP/32 pop-3 -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 imap -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 exec -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 login -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 syslog -S 0.0.0.0/0 -P udp
$IPFWADM -I $MODE deny -D $MYIP/32 shell -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 talk -S 0.0.0.0/0 -P udp
$IPFWADM -I $MODE deny -D $MYIP/32 ntalk -S 0.0.0.0/0 -P udp
$IPFWADM -I $MODE deny -D $MYIP/32 cfinger -S 0.0.0.0/0 -P tcp
$IPFWADM -I $MODE deny -D $MYIP/32 nfs -S 0.0.0.0/0 -P udp

# we stop all connections to our X server (if running)
# comment out the line below if you require X access

#$IPFWADM -I $MODE deny -D $MYIP/32 6000 -S 0.0.0.0/0 -P tcp

echo "firewall"
```



บรรณานุกรม

[1] วรวุฒิ เที่ยงธรรม, สันติ ศรีลาศักดิ์ : "*รู้จักลีนุกซ์*", บริษัท ออฟเซ็นเพรส จำกัด 2542

[2] สมศักดิ์ ลิมาวงษ์ปราณี : "*slackware linux ฉบับเพื่อการใช้งานจริง*",บริษัท ซัคเซสมีเดีย จำกัด

[3] Kai Hwang, Zniwei Xu : "*Scalable Parallel Computing*", McGrawHill 1999

[4] วิภา เพิ่มทรัพย์, สาโรจน์ ไพชยนต์ฤทธา : "*คู่มือติดตั้งแล้ใช้งาน Linux Red Hat 6.1*",บริษัท โปรวิชั่นจำกัด 2543

[5] *http://www.cacr.caltech.edu/beowulf/tutorial/building.html*

[6] *http://www.beowulf-underground.org/*

[7] *http://www.beowulf.org/*

[8] *http://metalab.unc.edu/LDP/HOWTO/mini/NFS-Root.html*

[9] *http://www.cacr.caltech.edu/beowulf/tutorial/beosoft/*

[10] *http://www-unix.mcs.anl.gov/mpi/mpich*

[11] *http://www.mpi.nd.edu/lam*

[12] *http://www.csse.monash.edu.au/~rajkumar/cluster/index.html*

[13] *http://www.cris.com/~rjbono/html/disklessboot.html*

[14] *http://www.beowulf.org/*

[15] *http://www-unix.mcs.anl.gov/mpi/*

[16] *http://www.fftw.org/*

[17] *http://www.beowulf-underground.org/doc_project/BIAA-HOWTO/Beowulf-Installation-and-Administration-HOWTO.html*

[18] *http://www-unix.mcs.anl.gov/mpi/mpich*

[19] *http://www.mpi.nd.edu/lam*

[20] *http://www.epm.ornl.gov/pvm/*

[21] *http://netlib.org/pvm3/book/pvm-book.html*

[22] *http://www.linuxdoc.org/HOWTO/NFS-HOWTO/*

[23] *http://yara.ecn.purdue.edu/~pplinux/PPHOWTO/pphowto-3.html#ss3.5*

[24] *http://www.itd.clrc.ac.uk/Publications/Cookbook/*

[25] *http://www.scriptics.com/*

[26] *http://www.sco.com/Technology/tcl/Tcl.html*

[27] *http://www.arsdigita.com/books/tcl/index*

[28] *http://hegel.ittc.ukans.edu/topics/tcltk/tutorial-noplugin/*

[29] *http://www.arsdigita.com/books/tcl/index*