โปรแกรมตรวจสอบแฟ้มข้อมูลบนลีนุกซ์

Linux File Integrity Checker

จัดทำโดย

นาย ธนาวุฒิ ธนามี

นาย อภิเชษฐ์ วิหคโต

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

โปรแกรมตรวจสอบแฟ้มข้อมูลบนลีนุกซ์

Linux File Integrity Checker

จัดทำโดย

นาย ธนาวุฒิ ธนามี

นาย อภิเชษฐ์ วิหคโต

อาจารย์ที่ปรึกษา

อาจารย์ ธนา หงษ์สุวรรณ

อาจารย์ อัครเดช วัชระภูพงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะ วิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมตรวจสอบแฟ้มข้อมูลบนลีนุกซ์

Linux File Integrity Checker

ผู้จัดทำ

1. นาย ธนาวุฒิ ธนามี รหัสประจำตัว 43015359
2. นาย อภิเชษฐ์ วิหคโต รหัสประจำตัว 43015397

.....อาจารย์ที่ปรึกษา

(อาจารย์ ธนา หงษ์สุวรรณ)

.....อาจารย์ที่ปรึกษา

(อาจารย์ อัครเดช วัชระภูพงษ์)

โปรแกรมตรวจสอบแฟ้มข้อมูลบนลีนุกซ์

นาย ธนาวุฒิ ธนามี
นาย อภิเชษฐ์ วิหคโต
อาจารย์ ธนา หงษ์สุวรรณ อาจารย์ที่ปรึกษา
อาจารย์ อัครเดช วัชระภูพงษ์ อาจารย์ที่ปรึกษา

บทคัดย่อ

ปัจจุบันระบบเครือข่ายมีความสำคัญมากขึ้น นำมาใช้ในการดำเนินงานธุรกรรมสำคัญๆ แต่การติดตั้งและดูแลระบบเครือข่ายให้มีความปลอดภัยต่อเหล่าอาชญากรทางอินเทอร์เน็ตนั้นยังคงเป็นไปอย่างไม่สมบูรณ์ โครงงานนี้เป็นโครงงานที่พัฒนาเพื่อศึกษาและสร้างตัวตรวจสอบแฟ้มข้อมูลบนลีนุกซ์เป็นการรักษาความปลอดภัยเชิงรับซึ่งลีนุกซ์จะมองทุกอย่าง เช่น อุปกรณ์ต่าง ๆ เป็นแฟ้มข้อมูล ดังนั้นจึงเน้นการรักษาความปลอดภัย ไปที่แฟ้มข้อมูลของระบบเป็นหลัก โดยยึดหลักการว่าการเปลี่ยนแปลงใด ๆ ที่เกิดขึ้นกับแฟ้มข้อมูลของระบบและไม่ได้เกิดจากการกระทำของตัวระบบเองแล้วมีความเสี่ยงที่จะมีการบุกรุกเกิดขึ้นได้ โดยการตรวจสอบจะใช้ MD5 มาช่วยในการตรวจสอบเพื่อแยกแยะการเปลี่ยนแปลงของแฟ้มข้อมูล

นอกจากนี้ยังตรวจสอบว่ามีการเข้าถึงแฟ้มข้อมูลมีการแก้ไขสิทธิของแฟ้มข้อมูล และมีการเปลี่ยนแปลงของไอโหนด ดังนั้นเมื่อมีการเปลี่ยนแปลง ณ. จุดใดจะเก็บข้อมูลของการเปลี่ยนแปลงนั้นๆ ไว้จึงง่ายต่อการที่ผู้ดูแลระบบจะตรวจสอบว่ามีการบุกรุกระบบนั้นๆหรือไม่

จุดสนใจของโครงงานนี้อยู่ที่ระบบแฟ้มข้อมูลและโพรเซสที่ทำงานในระบบแพลตฟอร์มลีนุกซ์และพัฒนาโดยภาษาซี

Linux File Integrity Checker

Mr. Thanawut Thanamee
Mr. Apichet Wihokto
Mr. Thana Hongsuwan Advisor
Mr. Akkradach Watcharapupong Advisor

ABSTRACT

Nowadays, the network system is used to run out many important businesses but the way to set up and maintain the network system to be more safety and can prevent the system is insecure and incomplete. This project is developed to create the Linux File Integrity Checker, which is a protective security. Because Linux looks everything as a file, the project will focus at a system file by look at all of the system files' changes. If the changes do not come from it's own system, that's mean there is a risk of attacking. The inspection use MD5 to distinguish the changes of system file.

In addition to check file access, file permission edition and I-node change. Every changes will be kept for checking later.

This project pays attention to the system file and the process that work in Linux platform and be developed with C language.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดี ด้วยความช่วยเหลือและได้รับความร่วมมือและได้รับคำปรึกษาจากอาจารย์ที่ปรึกษาทั้งสองท่านคือ อาจารย์ ธนา หงษ์สุวรรณและอาจารย์ อัครเดช วัชระภูพงษ์ ที่คอยให้คำแนะนำและเอาใส่ดูแลการทำโครงงานและตรวจสอบความถูกต้องของปริญญานิพนธ์ ซึ่งทางคณะผู้จัดทำต้องขอกราบขอบพระคุณเป็นอย่างสูง

นอกจากนี้ต้องขอกราบขอบคุณอาจารย์ประจำภาควิชา วิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่ช่วยประสิทธิ์ประสาทวิชาความรู้ให้แก่คณะผู้จัดทำ อีกทั้งภาควิชาวิศวกรรมคอมพิวเตอร์ที่เอื้อเฟื้ออุปกรณ์ สถานที่อำนวยความสะดวกสบาย ต่าง ๆ ในการทำโครงงานนี้

ขอบขอบคุณเพื่อน ๆ ห้องพี่ทุกคนที่เป็นกำลังใจให้ เพื่อนชาวไอแซ็ก (ISAG) ได้แก่ พี่ต่อ พี่เชาว์ พี่เดียร์ และอีกหลายคนที่ทำวิจัยในห้องไอแซ็กที่คอยช่วยเหลือเมื่อยามมีข้อบกพร่องและช่วยกันแก้ไข

สุดท้ายนี้ที่ขาดไม่ได้ขอขอบพระคุณบุคคลที่ทำให้เรามีวันนี้ คือ บิดา มารดาที่เคารพเป็นอย่างยิ่งที่ให้กำเนิดคนนี้ขึ้นมา ที่คอยให้กำลังใจ เป็นห่วงเป็นใย ให้คำสั่งสอน ให้การศึกษาอย่างสูงสุด พร้อมทั้งให้การสนับสนุนปัจจัยในด้านต่าง ๆ นับเป็นพระคุณอย่างยิ่งสุดที่จะหาอะไรมาเปรียบเทียบได้

คณะผู้จัดทำขอระลึกถึงพระคุณอันยิ่งใหญ่สุดที่จะประมาณกว่านี้ไว้กว่าชีวิตหาไม่และขอกราบพระคุณทุกท่านไว้ ณ ที่นี้ด้วย

นาย ธนาวุฒิ ธนามี
นาย อภิเชษฐ์ วิหคโต

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

การดำเนินงานธุรกรรมทางระบบเครือข่ายอินเทอร์เน็ตนั้น ต้องการความปลอดภัยสูง เพราะว่าการเปิดช่องว่างทำให้ผู้บุกรุกหรือแฮกเกอร์(Hacker) สามารถเข้ามาเปลี่ยนแปลงแฟ้มข้อมูลเพื่อผลประโยชน์ทางธุรกิจหรือเพื่อทำลายฐานข้อมูลที่สำคัญได้ ซอฟต์แวร์ตรวจสอบความมั่นคงระบบคอมพิวเตอร์จึงมีบทบาทมากขึ้น แต่ในปัจจุบันซอฟต์แวร์ประเภทนี้มีการพัฒนาน้อย ซึ่งการพัฒนาซอฟต์แวร์ส่วนมากเป็นการพัฒนาเชิงรุก (Active) มากกว่า แต่ก็ยังมีประสิทธิภาพไม่เพียงพอ ผู้บุกรุกยังสามารถเข้ามาในระบบเพื่อแก้ไขข้อมูลได้ โครงงานนี้จึงเป็นโครงงานที่จัดทำขึ้นเพื่อสร้างตัวตรวจสอบการเปลี่ยนแปลงของแฟ้มข้อมูลแบบเชิงรับ (Passive) เพื่อเป็นซอฟต์แวร์ที่มีความสามารถตรวจสอบการเปลี่ยนแปลงของแฟ้มข้อมูลภายในระบบ เป็นการตรวจสอบและป้องกันผู้บุกรุกที่สามารถผ่านการป้องกันเชิงรุกมาได้อีกชั้นหนึ่ง โดยโครงงานนี้มีเป้าหมายเพื่อสร้างซอฟต์แวร์ตรวจสอบการเปลี่ยนแปลงของแฟ้มข้อมูลที่พัฒนาโดยคนไทย ในระบบปฏิบัติการลีนุกซ์ และพัฒนาด้วยภาษาซี

หากเรามีกลไกการตรวจสอบการเปลี่ยนแปลงของแฟ้มข้อมูลที่ดีพอในทุกๆ จุดที่เกิดการเปลี่ยนแปลงแล้ว เราย่อมวิเคราะห์และแยกแยะได้ว่าการเปลี่ยนแปลงนั้นเกิดตามการใช้งานปกติหรือเกิดจากการกระทำของผู้บุกรุก จุดสนใจของโครงงานนี้อยู่ที่ระบบแฟ้มข้อมูลและโพรเซสที่ทำงานในระบบแพลตฟอร์มลีนุกซ์

1.2 วัตถุประสงค์ของโครงงาน

- เพื่อศึกษาการทำงานเครื่องตรวจสอบการเปลี่ยนแปลงของระบบแฟ้มข้อมูล
- เพื่อสร้างโปรแกรมต้นแบบสำหรับตรวจสอบการเปลี่ยนแปลงของระบบแฟ้มข้อมูลบนแพลตฟอร์มลีนุกซ์

1.3 ขอบเขตงานของโครงงาน

- โปรแกรมตรวจสอบการเปลี่ยนแปลงของระบบแฟ้มข้อมูลบนแพลตฟอร์มลีนุกซ์
- โปรแกรมที่ตรวจสอบการเปลี่ยนแปลงแฟ้มข้อมูลโดยใช้ MD5
- โปรแกรมที่สามารถตรวจสอบเวลาการเปลี่ยนแปลงในไอโหนด เวลาการเข้าถึงล่าสุด เวลาการแก้ไขล่าสุดและการแก้ไขสิทธิแฟ้มข้อมูลล่าสุด
- ตรวจสอบการเพิ่มหรือลบแฟ้มข้อมูลในระบบ
- พัฒนาเครื่องมือที่ใช้ง่ายในการตรวจสอบสำหรับผู้ดูแลระบบ
- สร้างโปรแกรมตรวจสอบการเปลี่ยนแปลงของระบบแฟ้มข้อมูลที่สำคัญ

1.4 ประโยชน์ที่คาดว่าจะได้รับ

- โปรแกรมที่สามารถติดตามการเปลี่ยนแปลงของแฟ้มข้อมูล
- เข้าใจการเขียนโปรแกรมด้วยภาษาซีบนแพลตฟอร์มลีนุกซ์มากยิ่งขึ้น
- ได้โปรโปรแกรมที่ตรวจสอบการเปลี่ยนแปลงแฟ้มข้อมูลที่ใช้งานได้งานง่ายต่อผู้ดูแลระบบ

บทที่ 2
ระบบไฟล์ในลีนุกซ์

ระบบไฟล์ของระบบลีนุกซ์นั้น สำหรับผู้ที่คุ้นเคยกับระบบไฟล์ในยูนิกซ์ก็สามารถเข้าใจได้ง่าย ส่วนผู้ที่ไม่คุ้นเคยก็ไม่ใช่เรื่องยาก ระบบลีนุกซ์นั้นนับว่าเป็นระบบปฏิบัติการยูนิกซ์ชนิดหนึ่ง ซึ่งมีการอ้างถึงทุกสิ่งทุกอย่างในระบบเป็นไฟล์ทั้งสิ้น ไม่ว่าเป็นอุปกรณ์ฮาร์ดแวร์ โปรแกรม หรืออะไรก็ตามแต่เนื่องจากปริมาณของไฟล์นั้นมีมาก ดังนั้นการจัดโครงสร้างของระบบไฟล์ในลีนุกซ์จึงเป็นสิ่งสำคัญ

2.1 ระบบไฟล์

ระบบไฟล์คือโครงสร้างการจัดเก็บข้อมูลในฮาร์ดดิสก์ เพื่อให้ระบบปฏิบัติการสามารถอ่านเขียนใช้งานไฟล์ที่ต้องการได้อย่างมีประสิทธิภาพ และควบคุมสิทธิในการเข้าใจของผู้ใช้ เช่น ในระบบปฏิบัติการดอส (Disk Operating System) ที่มีการใช้งานมาเป็นระยะเวลายาวนานมีระบบไฟล์ที่เรียกว่า FAT (File Allocation Table) ซึ่งมีทั้งแบบ 12 และ 16 บิต และสำหรับระบบไฟล์ในไมโครซอฟต์วินโดวส์ 95 ของไมโครซอฟต์ และ OS/2 ของไอบีเอ็มสนับสนุนระบบไฟล์ทั้งแบบ FAT12, FAT16 และ FAT32 แต่ระบบปฏิบัติการทั้งสองแบบเป็นระบบปฏิบัติการแบบผู้ใช้เดียว ทำให้ระบบไฟล์ไม่มีส่วนควบคุมสิทธิการเข้าใช้ แต่ระบบปฏิบัติการใหม่ๆ เช่น ไมโครซอฟต์วินโดวส์ เอ็นที จะใช้ระบบไฟล์ที่เรียกว่า NTFS (NT File System) หรือระบบปฏิบัติการ IBM OS/2 จะใช้ระบบไฟล์ที่เรียกว่า HPFS (High Performance File System) ซึ่งระบบไฟล์ทั้งสองจะมีส่วนที่ควบคุมสิทธิการเข้าใช้ของผู้ใช้ มีส่วนที่สนับสนุนการเข้าใช้จากผู้ใช้หลายคน

ระบบปฏิบัติการ	ระบบไฟล์
DOS	FAT12, FAT16
MS Windows 95	FAT12, FAT16
MS Windows 95, MS Windows 98, MS Windows ME	FAT12, FAT16, FAT32
MS Windows NT, MS Windows 2000	FAT12, FAT16, NTFS, HPFS
IBM OS/2	FAT12, FAT16, HPFS
Linux	FAT12, FAT16, FAT32, HPFS, NTFS, Ext1, Ext2, UFS, VFAT, Minix, ISOFS, HFS, AFFS, ADFS, SYSV

ตารางที่ 2-1 ระบบไฟล์ของระบบปฏิบัติการชนิดต่างๆ

2.2 ระบบชื่อไฟล์ในระบบลีนุกซ์

ระบบการตั้งชื่อไฟล์ในระบบปฏิบัติการระบบลีนุกซ์นั้นต่างจากดอสคือ ดอสจะให้มีการตั้งชื่อไฟล์ด้วยชื่อขนาด 8 ตัวอักษร และสกุลอีก 3 ตัวอักษร ส่วนระบบไฟล์ของระบบลีนุกซ์ในรุ่นแรกๆ นั้นจะใช้ระบบไฟล์ของ Minix (Minix File System) ซึ่งเป็นระบบไฟล์ที่ถูกใช้ในระบบปฏิบัติการ Minix นี้จะมีขีดความสามารถจำกัดในการเก็บชื่อไฟล์ได้สูงสุดเพียง 14 ตัวและมีข้อจำกัดมากมายในระบบไฟล์นี้ต่อมาได้มีการปรับปรุงและพัฒนาเพิ่มเพื่อให้สามารถเก็บชื่อไฟล์ที่มีขนาดเพิ่มขึ้นได้ถึง 30 ตัวอักษรสำหรับลีนุกซ์ในปัจจุบันนี้ใช้ระบบไฟล์ที่เรียกว่า Ext2 (Extended Files System 2) ซึ่งสามารถตั้งชื่อไฟล์ได้ถึง 255 ตัวอักษร

การตั้งชื่อไฟล์ในระบบลีนุกซ์นั้นสามารถใช้ตัวอักษร ตัวเลข ขีดเส้นใต้ และยังให้ความแตกต่างกันระหว่างอักษรเล็กกับอักษรตัวใหญ่ในรูปแบบที่เรียกว่า เคสเซนซิทีฟ (Case Sensitive) ด้วย เช่น FILE1, file1 และ File1 นั้นชื่อไฟล์ทั้ง 3 มีความแตกต่างกัน

สำหรับการตั้งชื่อไฟล์นั้นควรหลีกเลี่ยงการใช้เครื่องหมายพิเศษต่างๆ เช่น ^ " ' , - ? [ ] ( ) ~ ! $ { } < > # @ & / และยิ่งกว่านั้นถ้าไฟล์ใดที่มีชื่อขึ้นต้นด้วย " . " (dot) จะกลายเป็นไฟล์ที่ถูกซ่อนไว้ (hidden file) ซึ่งจะไม่แสดงรายชื่อไฟล์ออกมาให้เห็น หากต้องการเรียกไฟล์ที่ถูกซ่อนขึ้นมาดูต้องเพิ่มพารามิเตอร์พิเศษให้กับคำสั่งในการขอดูรายชื่อไฟล์

ระบบไฟล์	ขนาดของชื่อไฟล์(ตัวอักษร)	ขนาดไฟล์ (เมกกะไบต์)
Minix File System	14	64
Extended File System(Ext1)	255	2048
Extended File System(Ext2)	255	2048

ตารางที่ 2-2 ระบบชื่อและขนาดของไฟล์ในรูปแบบต่างๆ

2.3 ระบบไฟล์ในลีนุกซ์

ขนาดของไฟล์ในระบบลีนุกซ์รุ่นแรกๆ นั้นไม่สามารถเก็บไฟล์ที่มีขนาดใหญ่เกินกว่า 64 เมกะไบต์ได้ทำให้ผู้ใช้งานที่ต้องการจะใช้ไฟล์ขนาดใหญ่บนลีนุกซ์ไม่สามารถทำได้ จึงได้มีการออกแบบและสร้างระบบไฟล์ขึ้นใหม่เรียกว่า Ext (Extended File System) เมื่อเดือนเมษายน ปี 2535 เป็นระบบไฟล์ที่สองที่ระบบปฏิบัติการลีนุกซ์สนับสนุนการทำงานด้วย โดยระบบไฟล์นี้ถูกสร้างโดย "เรมี คาร์ด" (Remy Card) เพื่อให้ระบบสามารถสนับสนุนการใช้งานไฟล์ขนาดใหญ่ โดยระบบไฟล์นี้มีพื้นฐานมาจากระบบไฟล์ใน Minix ทำให้มีข้อจำกัดในระบบไฟล์ของระบบลีนุกซ์ลดลง แต่ผลจากการออกแบบส่วนจัดการพื้นที่ว่างที่ยังไม่ดีพอทำให้ระบบจะเกิดพื้นที่ว่างที่ไม่ต่อเนื่องกันได้ง่ายทำให้เกิดปัญหาที่เรียกว่าแฟรก

เมนต์เตชั่น(fragmentation) ส่งผลให้ระบบทำงานช้ากว่าระบบไฟล์ของ Minix ที่เป็นระบบไฟล์ต้นแบบต่อมา "เรมี คาร์ด" และ "เวย์เน่ ดาวิสัน" (Wayne Davison) ได้ทำมีการออกแบบระบบไฟล์ใหม่เรียกว่าระบบไฟล์แบบ Ext2 (Extended Files System 2) ในเดือนมกราคม ปี 2536 หลังจากที่เก็บข้อมูลความต้องการจากผู้ใช้ที่ต้องการระบบไฟล์ที่มีประสิทธิภาพ เป้าหมายในการออกแบบระบบไฟล์นี้จึงมุ่งไปที่ประสิทธิภาพและความถูกต้องของข้อมูล และสามารถรองรับขนาดของพาร์ทิชันที่สูงถึง 4 เทราไบต์ ระบบไฟล์นี้มีพื้นฐานมาจากระบบไฟล์ Ext ระบบไฟล์นี้เป็นที่ยอมรับใช้งานจนถึงปัจจุบัน

2.4 การจัดแบ่งพาร์ทิชันของระบบไฟล์

ในการติดตั้งระบบปฏิบัติการเรดแฮ็ตลีนุกซ์แบบเซิร์ฟเวอร์ (Server) โปรแกรมการติดตั้งจะสร้างพาร์ทิชันเพื่อที่จะใช้ในการติดตั้งระบบปฏิบัติการลีนุกซ์จะแตกต่างกับระบบปฏิบัติการดอสและวินโดว์ที่เราคุ้นเคย ด้วยระบบปฏิบัติการดอสและวินโดว์จะใช้วิธีการอ้างถึงพาร์ทิชันต่างๆ โดยการอาศัยตัวอักษรประจำพาร์ทิชัน เช่น C:,D: หรือ F: เป็นต้น แต่ในระบบปฏิบัติการลีนุกซ์ จะมองระบบไฟล์ที่จัดเก็บไว้ในแต่ละพาร์ทิชันต่อเนื่องกันตลอดทุกพาร์ทิชัน โดยเริ่มจากไดเรกทอรีราก (root directory) ต่อเนื่องไปยังไดเรกทอรีต่างๆ ที่ถูกจัดเก็บไว้ในแต่ละพาร์ทิชัน แต่อย่างไรก็ตามระบบปฏิบัติการจะไม่ได้มองพื้นที่ว่างทั้งหมดในแต่ละพาร์ทิชันเป็นพื้นที่ว่างผืนเดียวกันหมด

การจัดระบบไฟล์ของลีนุกซ์แบบนี้ทำให้การจัดเก็บไฟล์ในแต่ละไดเรกทอรีขึ้นกับว่าไดเรกทอรีนั้นเก็บในพาร์ทิชันใด ดังนั้นไฟล์ที่จัดเก็บจึงต้องเก็บในพาร์ทิชันเดียวกันด้วย ผลจากการจัดการไฟล์ด้วยวิธีการนี้จึงต้องมีการกำหนดจุดเชื่อมต่อ (Mount Point) ให้กับแต่ละพาร์ทิชันต่างๆ เพื่อให้เข้าถึงพาร์ทิชันทั้งหมดได้

ไดเรกทอรีรากจะเก็บไฟล์โปรแกรมและไฟล์คุณสมบัติที่จำเป็นในการบูตระบบ และไดเรกทอรีที่ใช้ในการเชื่อมต่อกับพาร์ทิชันอื่นๆ ดังนั้นพาร์ทิชันหลัก (root partition หรือ main partition) จะถูกเชื่อมต่อเข้ากับระบบปฏิบัติการลีนุกซ์เป็นลำดับแรก ถ้าพาร์ทิชันหลักเสียหายจะทำให้ไม่สามารถบูตหรือเริ่มต้นการทำงานของระบบปฏิบัติการได้ ด้วยวิธีการนี้ระบบไฟล์ของระบบลีนุกซ์จึงมีเพียงหนึ่งไดเรกทอรีแต่สามารถทำงานบนระบบที่มีฮาร์ดดิสก์กี่ตัวก็ได้

การที่ระบบปฏิบัติการลีนุกซ์ทำการแยกย่อยฮาร์ดดิสก์ออกเป็นหลายๆ พาร์ทิชันแทนการให้พาร์ทิชันขนาดใหญ่เพียงหนึ่งหรือสองพาร์ทิชัน เพราะต้องการให้แต่ละพาร์ทิชันมีการจองขนาดการใช้พื้นที่ใช้สอยไว้ล่วงหน้า เพื่อป้องกันการใช้พื้นที่ว่างจนหมดแล้วทำให้ระบบปฏิบัติการไม่สามารถทำงานต่อไปได้ ยกตัวอย่างเช่น ถ้ามีการติดตั้งโปรแกรมประยุกต์จนเต็มพื้นที่ใช้สอยทั้งหมดจะทำให้ระบบปฏิบัติการลีนุกซ์ไม่สามารถทำงานต่อไปได้

การจัดเก็บไฟล์ในลีนุกซ์จะเก็บในรูปของบล็อก (Block) โดยจะมีขนาดเป็นจำนวนเท่าของ 512 ไบต์ ขึ้นอยู่กับการกำหนดขนาดของระบบนั้นๆ ซึ่งการกำหนดขนาดเล็กหรือใหญ่ก็มีข้อดี ข้อเสียแตกต่างกันไป หากบล็อกมีขนาดใหญ่จะทำให้การแลกเปลี่ยนข้อมูลระหว่างดิสก์กับหน่วยความจำทำได้รวดเร็ว

แต่อาจจะทำให้เกิดการสูญเสียเนื้อที่ (fragmentation) ไปได้โดยง่ายการจัดเก็บโดยปกติจะเก็บเป็นบล็อกที่ติดต่อกัน แต่ถ้ามีไฟล์อื่นมาแทรกก็สามารถกระโดดข้ามบล็อกนั้นไปได้ เนื่องจากมีตารางบอกว่าไฟล์นั้นเก็บข้อมูลอยู่ที่บล็อกใดบ้างโดยเก็บไว้ในส่วนของไอโหนด (inode) ซึ่งจะกล่าวในลำดับต่อไป

2.5 การจัดไดเรกทอรีของระบบไฟล์ลีนุกซ์

ลีนุกซ์นั้นใช้โครงสร้างของไฟล์และไดเรกทอรีแบบของ FSSTND (Linux File System Standard) ซึ่งต่อมาได้มีระบบหลายระบบนำรูปแบบการจัดการไฟล์และไดเรกทอรีแบบนี้ไปใช้และพัฒนาเป็นระบบโครงสร้างใหม่ที่เรียกว่า FHS (Filesystem Hierarchy Standard) แทน โดยมีการจัดเป็นแบบลำดับชั้น โดยมีลักษณะคล้ายกับต้นไม้กลับหัว โดยจุดเริ่มต้นหรือชั้นแรกจะเรียกว่าเป็นราก (root) ซึ่งจะเขียนแทนด้วยเครื่องหมาย / ไฟล์แต่ละไฟล์นั้นอาจจะเป็นข้อมูลที่สร้างขึ้น หรือโปรแกรมก็ได้ หรือแม้แต่ใช้เป็นที่เก็บไฟล์เองก็ได้ ซึ่งไฟล์ลักษณะนี้จะจัดว่าเป็นไฟล์ประเภทไดเรกทอรีรูปแบบการจัดเก็บไฟล์แบบลำดับชั้นนี้จะทำให้การจัดเป็นไฟล์เป็นระบบ ทำให้ง่ายต่อการใช้งานและดูแลโครงสร้างหลักของการจัดระบบไฟล์

2.6 ไอโหนด (Inode)

เคอร์เนลจะไม่สามารถติดต่อหรือรู้จักไฟล์โดยตรงแต่เคอร์เนลจะรับรู้สิ่งต่างๆ เกี่ยวกับไฟล์ที่มีอยู่ได้จากโครงสร้างข้อมูลที่เรียกว่าไอโหนด โดยไอโหนดจะประกอบด้วยข้อมูลต่างๆ เกี่ยวกับไฟล์เป็นจำนวนมาก เช่น การแสดงสิทธิต่างๆ ของไฟล์

- การบอกถึงชนิดของไฟล์
- แสดงถึงเจ้าของ และกลุ่มของเจ้าของไฟล์
- วัน เวลา ที่สร้างไฟล์ เปลี่ยนแปลงไฟล์ หรือแอกเซสไฟล์
- จำนวนลิงค์ที่ไฟล์เชื่อมต่อด้วย
- เก็บตำแหน่งที่อยู่ของข้อมูลในไฟล์

ไอโหนดนั้นจะถูกสร้างขึ้นมาโดยอัตโนมัติตอนสร้างระบบไฟล์ และถ้าหากไอโหนดนั้นเต็มจะไม่สามารถสร้างไฟล์ขึ้นมาได้เลย ถึงแม้ว่าจะยังมีเนื้อที่ว่างอยู่

2.7 ประเภทของไฟล์

ไฟล์ในระบบลีนุกซ์ได้มีการแบ่งประเภทออกไปหลายรูปแบบ และรูปแบบนั้นยังขึ้นอยู่กับกลุ่มของผู้สร้างลีนุกซ์ด้วย

- ไฟล์ทั่วไป (Regular file)
- ไดเรกทอรี (Directories)
- ไฟล์พิเศษ (Character and Block device file)
- ไฟล์ที่เชื่อมต่อ (Link File)

2.7.1 ไฟล์ทั่วไป (Regular file)

เป็นไฟล์ธรรมดาที่อาจสร้างขึ้นเองจากเอดิเตอร์ หรืออาจจะทำการสำเนามาจากไฟล์อื่น หรืออาจจะเป็นโปรแกรมใช้งานต่างๆ ก็ได้ โดยลักษณะการรายงานละเอียดต่างๆ ของไฟล์ เช่น สิทธิไฟล์ เจ้าของไฟล์ กลุ่มเจ้าของไฟล์ ขนาดหรือวันเวลาที่ทำการสร้าง เปลี่ยนแปลงแก้ไขไฟล์ ซึ่งจะได้ข้อมูลต่างๆ นี้มาจากไอโหนด (ซึ่งจะกล่าวในลำดับต่อไป) นั่นเอง

```
[arnut@database / sbin]$ ls -al | more
total 4137
drwxr - xr -x 3 root root 2048 Oct 4 11:34 .
drwxr - xr -x 17 root root 1024 Oct 2 03:52 ..
-rwxr - xr -x 1 root root 29544 Jun 18 22:05 arp
-rwxr - xr -x 1 root root 8244 Mar 22 1999 badblocks
-rwsr - xr -x 1 root root 10708 Jun 2 19:51 cardctl
-rwxr - xr -x 1 root root 35096 Jun 2 19:51 cardmgr
-rwxr - xr -x 1 root root 56728 Apr 17 1999 cfdisk
-rwxr - xr -x 1 root root 21192 Apr 19 1999 chkconfig
-rwsr - sr -x 1 root root 147 Aug 16 11:05 chkyppasswdd
-rwxr - xr -x 1 root root 4276 Apr 17 1999 ctrlaltdel
-rwxr - xr -x 1 root root 33364 Mar 22 1999 debugfs
-rwxr - xr -x 1 root root 27316 Apr 20 1999 depmod
-rwxr - xr -x 1 root root 6620 Mar 22 1999 dumpe2fs
-rwxr - xr -x 1 root root 410716 Mar 22 1999 e2fsck
-rwxr - xr -x 1 root root 4916 Mar 22 1999 e2label
-rwxr - xr -x 1 root root 93980 Apr 17 1999 fdisk
--More--
```

2.7.2 ไดเร็กทอรี (Directory)

เป็นไฟล์ชนิดหนึ่งซึ่งใช้เก็บบรรดาไฟล์ต่างๆ หรือไดเรกทอรีด้วยกันเอง ปริมาณไฟล์ในระบบลีนุกซ์ นั้นมีจำนวนมาก จึงจำเป็นต้องมีรูปแบบการจัดเก็บไฟล์ให้อยู่ในไดเรกทอรี่ที่เหมาะสม เพื่อให้ผู้ใช้สามารถทำนายได้ว่าไฟล์ควรอยู่ในไดเร็กทอรีใด

2.7.3 ไฟล์พิเศษ(Block&Character device files)

ลีนุกซ์มีไฟล์ชนิดพิเศษอยู่ 2 ชนิดคือ แบบบล็อก (block device file) และแบบตัวอักษร (Character device file) โดยลักษณะความแตกต่างกันของแบบบล็อกกับแบบคาเรกเตอร์ คือ ลักษณะการรับส่งข้อมูล ซึ่งถ้าเป็นไฟล์แบบตัวอักษรจะมีการรับส่งทีละ 1 ตัวอักษร การส่งแบบบล็อกนั้นจะมีการรับส่งข้อมูลทีละบล็อก ซึ่งอาจเป็น 512, 1024 หรือ 2048 ตัวอักษรก็ได้ ไฟล์ชนิดพิเศษเหล่านี้จะอยู่ในไดเรกทอรี “dev” ซึ่งเป็นไดเรกทอรีที่เก็บอุปกรณ์ต่างๆ ของระบบ เช่น ฮาร์ดดิสก์ ดิสก์ไดรฟ์ จอภาพ และ หน่วยความจำ เป็นต้น

2.7.4 ไฟล์ที่เชื่อมต่อ (Link Files)

ลักษณะการเชื่อมต่อไฟล์นั้นมีอยู่ 2 แบบคือการเชื่อมต่อทางกายภาพ (Hard Link) กับการเชื่อมต่อโดยนัย (Symbolic Link)

2.7.4.1 การเชื่อมต่อกายภาพ (Hard Link) การเชื่อมต่อแบบนี้จะมีการใช้ไอโหนด เดียวกันกับไฟล์ต้นฉบับ เสมือนกับการสร้างไฟล์ใหม่ขึ้นมา เพียงแต่ค่าไอโหนดนั้นซ้ำกับของเดิม และ ไอโหนด จะมีตัวนับจำนวนไฟล์ที่เชื่อมต่อด้วย

2.7.4.2 การเชื่อมต่อโดยนัย (Symbolic Link) เป็นลักษณะการเชื่อมต่อเช่นเดียวกันแต่จะมีการสร้างไอโหนด ของไฟล์ตัวเองขึ้นมาใหม่ โดยไฟล์ใหม่ที่ทำการสร้างขึ้นจะเก็บไดเร็กทอรีและชื่อไฟล์ที่ต้องการเชื่อมต่อด้วย (เหมือนกับการสร้าง Shortcut ใน Windows) หากมีการเปลี่ยนแปลงไฟล์ต้นฉบับจะส่งผลกระทบต่อไฟล์ที่เชื่อมต่อด้วย

2.8 การจัดการไฟล์และไดเร็กทอรี

- **การคัดลอก (COPY)**

  เราสามารถทำการสำเนาแฟ้มข้อมูล (หรืออาจจะเป็นไดเรกทอรีด้วยก็ได้) โดยใช้คำสั่ง

  # cp file1 file2

  คำสั่งข้างบนจะทำการสำเนาแฟ้มข้อมูล (copy) จาก file ไปเป็นชื่อ file2

- **การลบ (REMOVE)**

  หากต้องการทำการลบแฟ้มข้อมูล (หรืออาจจะเป็นไดเรกทอรี) ก็สามารถใช้คำสั่ง

  # rm filename

- **การย้าย/เปลี่ยนชื่อ (MOVE)**

  เราสามารถใช้คำสั่ง mv เพื่อทำการย้ายแฟ้มจากไดเรกทอรีหนึ่ง ไปสู่อีกไดเรกทอรีหนึ่ง

  # mv filename directory

  คำสั่งข้างบนจะทำการย้ายแฟ้มชื่อ filename จากไดเรกทอรีปัจจุบัน ไปสู่ไดเรกทอรีที่ ชื่อdirectory นอกจากนี้เราสามารถใช้คำสั่ง mv เพื่อใช้ทำการเปลี่ยนชื่อไฟล์ได้

  # mv file1 file2

  คำสั่งนี้จะทำการเปลี่ยนชื่อจาก file1 ไปเป็นชื่อ file2

- **การ mount ระบบไฟล์**

  การ mount จะเป็นการนำระบบไฟล์เข้าไปเชื่อมต่อกับระบบไฟล์เดิม โดยการนำระบบไฟล์ที่ต้องการเชื่อมต่อมาทับตรงจุดที่ต้องการเชื่อม ซึ่งเรียกว่า mount point หลังจากทำการเชื่อมต่อแล้วระบบไฟล์ที่นำมา mount นั้นจะเป็นเสมือนส่วนหนึ่งของระบบไฟล์ การ mount ในระบบลีนุกซ์นั้นจะอนุญาตเฉพาะผู้ดูแลระบบและนอกจากการ mount ระบบไฟล์แล้วยังสามารถ mount อุปกรณ์ให้เป็นเสมือนกับไฟล์หนึ่งในระบบก็กระทำได้อย่างเช่น

  การ Mount CD-ROM

  # mount /dev/cdrom /mnt/cdrom

  การ Mount Floppy Disk

  # mount /dev/fd0 /mnt/floppy

- **การยกเลิกการ Mount**

  สำหรับการยกเลิกการ Mount นั้นสามารถยกเลิกโดยใช้คำสั่ง unmount

  # cd /root

  # umount /dev/cdrom

- **การกำหนดสิทธิการใช้งานไฟล์ (File Permission)**

  ลีนุกซ์เป็นระบบปฏิบัติการที่มีการใช้งานไฟล์ต่างๆ ร่วมกันในกลุ่มของผู้ใช้งาน หากให้สิทธิแก่ผู้ใช้ทั้งหมดในการจัดการหรือกระทำใดได้กับไฟล์นั้นย่อมก่อให้เกิดความวุ่นวายและสับสนได้ ดังนั้นลีนุกซ์จึงได้มีการจำกัดสิทธิการใช้งานไฟล์โดยกำหนดประเภทของผู้ใช้งานไฟล์ออกเป็น 3 กลุ่มใหญ่ๆ คือ

  - เจ้าของไฟล์หรือผู้สร้างไฟล์ (owner)
  - ผู้ใช้ในกลุ่มเดียวกับเจ้าของไฟล์ (group)

- บุคคลอื่น (other or universe)

สิทธิในไฟล์นั้นประกอบไปด้วยการให้สิทธิในการอ่านไฟล์ "r" หรือ "read" เขียนหรือแก้ไขเปลี่ยนแปลงไฟล์ "w" หรือ "write" และเรียกปฏิบัติงานหรือค้นหา "x" หรือ "execute" , "search" และการไม่มีสิทธิจะใช้สัญลักษณ์ขีด "-"

รูปแบบการแสดงสิทธิในไฟล์นั้นจะใช้ตัวเลข 9 บิต หรือเรียกว่า nine permission bits โดย 3 บิตแรกเป็นการแสดงสิทธิของเจ้าของไฟล์ 3 บิตถัดมานั้นเป็นการแสดงสิทธิของกลุ่มเจ้าของไฟล์และ 3 บิตสุดท้ายเป็นการแสดงสิทธิของบุคคลอื่น ดังตัวอย่างในตารางที่ 2-4

สิทธิ์การใช้งาน	รหัส
-rwxrwxrwx	777
-rwxrwx---	770
-rwx------	700
-rw-rw-rw-	666
-rw-rw----	660
-rw-------	600
-r--------	400

ตารางที่ 2-3 สิทธิในการเข้าใช้งาน

ตัวอย่างเช่น เมื่อใช้คำสั่งขอดูไฟล์ชื่อ file1 จะพบรหัสแสดงสิทธิดังนี้

```
[arnut@database arnut]$ ls -l file1
-rw-r-  -r-  -  1  arnut  lecturer  0  Oct  22  17:58  file1
[arnut@database arnut]$
```

ตำแหน่งแรกที่ปรากฏทางซ้ายของบรรทัดนั้นใช้บ่งประเภทของไฟล์ในที่นี้เป็นรหัส "-" หมายถึงไฟล์ทั่วไป ค่าสิทธิของเจ้าของแฟ้มเป็น "rw-" คืออ่านและแก้ไขไฟล์ได้ ส่วนผู้ใช้ในกลุ่มเจ้าของไฟล์และบุคคลอื่นนั้นมีสิทธิเป็น " r- - " คือสามารถอ่านไฟล์ได้อย่างเดียวนั้น

การใช้งาน	สิทธิ์ขั้นต่ำ	ตัวอย่างการใช้งาน
พิมพ์ไฟล์	-r	$ cat sample_file
แก้ไขไฟล์	-rw	$ vi sample_file
ลบไฟล์	-w	$ rm sample_file
เพิ่มข้อมูล	-w	$ cat >> sample_file
เรียกใช้โปรแกรมไบนารี่	-x	$ run binary
เรียกใช้สคริปต์	-rw	$ run script

ตารางที่ 2-4 สิทธิขั้นต่ำในการใช้งานไฟล์

- การเปลี่ยนแปลงสิทธิต่างๆ ในไฟล์

การเปลี่ยนแปลงสิทธิต่างๆ ในไฟล์จะทำได้นั้นมีเพียง 2 กรณีคือผู้ที่ทำการเปลี่ยนแปลงนั้นเป็นเจ้าของไฟล์หรือเป็นผู้ดูแลระบบ การเปลี่ยนแปลงสิทธิต่างๆ ของไฟล์สามารถกระทำได้โดยใช้คำสั่ง chmod ตามด้วยสิทธิที่ต้องการให้เป็นและต่อท้ายด้วยชื่อไฟล์ที่ต้องการ

การเปลี่ยนแปลงสิทธิโดยให้ผู้ใช้ทุกกลุ่มนั้นสามารถใช้คำสั่ง chmod ตามด้วยเลขฐานแปดหรือกลุ่มตัวอักษรก็ได้ดังตารางที่ 4.5

กลุ่มผู้ใช้	เครื่องหมายการจัดการ	รูปแบบของสิทธิ์
-u (เจ้าของไฟล์)	+ (เพิ่มสิทธิ์)	-r(การอ่าน)
-g (บุคคลในกลุ่มเจ้าของไฟล์)	= (กำหนดสิทธิ์)	-w(การเขียน)
-o(บุคลทั่วไป)	- (ลดสิทธิ์)	-x(การเรียกใช้หรือค้นหา)
-a(ทุกคนทุกกลุ่ม)		

ตารางที่ 2-5 การเปลี่ยนแปลงสิทธิโดยใช้ตัวอักษร

และการกำหนดสิทธิในส่วนของกลุ่มผู้ใช้งานและรูปแบบของสิทธินั้น เราสามารถกำหนดได้มากกว่าหนึ่งกลุ่มได้ เช่น g+w จะหมายถึงกำหนดเพิ่มให้บุคคลในกลุ่มเจ้าของไฟล์สามารถแก้ไขไฟล์ได้

```
[arnut@database arnut]$ Is - I test_sql
-rw-r- - r- - 1 arnut lecturer 0 Oct 22 17:58 test_sql1
[arnut@database arnut]$ chmod g+w test_sql
[arnut@database arnut]$ Is - I test_sql
-rw-rw-r- - 1 arnut lecturer 0 Oct 22 17:58 test_sql
[arnut@database arnut]$
```

นอกจากนี้ยังสามารถเปลี่ยนผู้เป็นเจ้าของไฟล์ และกลุ่มของเจ้าของไฟล์ได้เช่นกัน โดยใช้คำสั่ง chown (change owner) และ chgrp (change group)

- การกำหนดสิทธิต่างๆ ในไฟล์โดยอัตโนมัติ

ในเชลล์จะมีคำสั่ง umask ไว้กำหนดค่าสิทธิต่างๆ ในไฟล์โดยอัตโนมัติเก็บไว้ที่ .profile สำหรับบอร์นเชลล์และคอร์นเชลล์ หรือ .chrc สำหรับซีเชลล์ ชื่อไฟล์ทั้งสองนี้เป็นไฟล์เริ่มต้นในการใช้งานทุกครั้ง

การกำหนดเช่นนี้เพื่อที่จะได้ไม่ต้องทำการกำหนดสิทธิหรือเปลี่ยนแปลงสิทธิของไฟล์ทุกครั้งที่มีการสร้างขึ้นใหม่ โดยรูปแบบของคำสั่ง umask จะตามด้วยตัวเลขฐานแปดจำนวน 3 หลัก ซึ่งตัวเลขจะแทนสิทธิต่างๆ ในทางตรงข้ามกับการกำหนดสิทธิของไฟล์ ดังตารางที่ 2 – 6

โดยทั่วไปแล้ว ค่าของ umask จะถูกกำหนดให้เป็น 022 คือให้เจ้าของไฟล์นั้นมีสิทธิทุกอย่างในการดำเนินการกับไฟล์ ส่วนสมาชิกในกลุ่มและผู้ใช้อื่นนั้นกำหนดให้เพียงสามารถอ่านและเอ็กซีคิวต์ไฟล์ได้เท่านั้น

umask	**สิทธิของไฟล์**	**สิทธิของไดเรกทอรี**
077	600(rw - - - - - - -)	700 (rwx - - - - -)
067	600 (rw - - - - - - -)	710 (rwx - -x - - -)
066	600 (rw - - - - - - -)	711 (rwx - -x - - x)
027	640 (rw - r - - - - -)	750 (rwxr - x - - -)
026	640 (rw - r - - - - -)	751 (rwxr - x - - x)
022	644 (rw - r - - r - -)	755 (rwxr - x r - x)
000	666 (rw - r w - r w -)	777 (rwxrwxrwx)

ตารางที่ 2-6 การกำหนดสิทธิของไฟล์และไดเรกทอรีโดยใช้ umask

2.9 ไฟล์สำคัญในการใช้งาน

2.9.1 ไฟล์ที่มองไม่เห็น (Hidden Files)

ในระบบปฏิบัติการลีนุกซ์ ชื่อไฟล์ และไดเร็กทอรีที่ขึ้นต้นด้วยจุด (.) ระบบจะถือว่าเป็นไฟล์ชนิดพิเศษโดยจะไม่สามารถมองเห็นได้จากมุมมองปกติ เป็น ไฟล์ซ่อน (Hidden Files) จะต้องทำการกำหนดออปชั่นพิเศษเพื่อทำการแสดงรายชื่อไฟล์ เช่น ls -a

2.9.2 ไฟล์สำคัญในระดับผู้ใช้

เมื่อผู้ใช้แต่ละคนล็อกอินเพื่อเข้าใช้ระบบ จะมีไฟล์สำคัญหลายๆ ไฟล์ที่เกี่ยวข้องกับการทำงานของผู้ใช้แต่ละคน โดยไฟล์เหล่านี้จะมีอยู่ในโฮมไดเรกทอรีของผู้ใช้แต่ละคน เพื่อเหตุผลในการทำงานต่างๆ ซึ่งมีรายการดังนี้

ไฟล์ .bash_history	ทำหน้าที่เก็บรายการคำสั่ง ที่ผู้ใช้ทำการเรียกใช้เอาไว้ ทำให้สามารถเรียกใช้คำสั่งเก่าได้
ไฟล์ .bashrc	จะถูกประมวลผลทุกครั้ง ที่ผู้ใช้ทำการสร้างเชลล์ขึ้นมาใหม่หรือเรียกใช้คำสั่งที่ไม่ใช่คำสั่งภายในของเชลล์
ไฟล์ .bash_profile	จะถูกประมวลผลทุกครั้ง เมื่อผู้ใช้ทำการ login เพื่อเข้าใช้ระบบ
ไฟล์ .bash_logout	จะถูกประมวลผลทุกครั้ง เมื่อผู้ใช้ทำการ logout เพื่อออกจากระบบ

ตารางที่ 2-7 ไฟล์สำคัญในระดับผู้ใช้

- ไฟล์สำคัญระดับการใช้งาน

ไฟล์ /etc/lilo.conf	เป็นไฟล์ที่เก็บค่าคอนฟิกูเรชั่นของโปรแกรม Lolo (Linux Loader)
ไฟล์ /etc/inittab	เป็นไฟล์ที่เก็บค่าคอนฟิกูเรชั่นของการเริ่มงานระบบ
ไฟล์ /etc/profile	เป็นไฟล์ที่เก็บคำสั่งเริ่มต้นของระบบที่ต้องการสั่งให้ทำงานตอนบู๊ตระบบ
ไฟล์ /etc/fstab	เป็นไฟล์ที่เก็บค่าคอนฟิกูเรชั่นของการเชื่อมต่อระบบไฟล์ในขณะบู๊ตระบบ
ไฟล์ /tec/X11/XF86Config-4	เป็นไฟล์ที่เก็บค่าคอนฟิกูเรชั่นของระบบ X Windows

ตารางที่ 2-8 ไฟล์สำคัญระดับใช้งาน

2.10 สรุป

ในบทนี้เราก็ได้เรียนรู้เกี่ยวกับระบบไฟล์ของระบบปฏิบัติการลีนุกซ์ ไม่ว่าจะเป็นระบบไดเรกทอรีทั้งหมดที่มีในลีนุกซ์, ประเภทของระบบไฟล์, การจัดการกับระบบไฟล์ รวมทั้งการกำหนดสิทธิการเข้าใช้งานไฟล์ ซึ่งถือเป็นเรื่องสำคัญสำหรับผู้ที่ต้องการศึกษาระบบปฏิบัติการลีนุกซ์ ต้องเรียนรู้การใช้งานระบบไฟล์อย่างถูกต้องเพื่อความเสถียรภาพของระบบและนำไปใช้ในส่วนอื่นๆ

บทที่ 3
การทำงานกับไฟล์

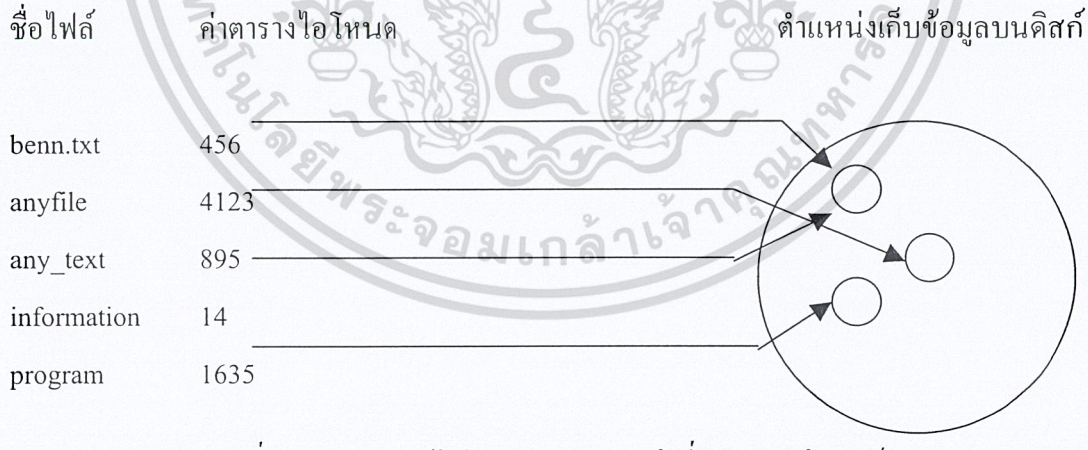
3.1 โครงสร้างของไฟล์บนระบบยูนิกซ์

ไฟล์ในระบบยูนิกซ์เป็นสิ่งที่มีความสำคัญ เนื่องจากมีการทำงานกันอย่างใกล้ชิดระหว่างระบบปฏิบัติการ, อุปกรณ์และ ระบบไฟล์ และที่สำคัญยูนิกซ์มองทุกสิ่งทุกอย่างเป็นไฟล์ โดยโปรแกรมสามารถใช้งานดิสก์ พอร์ตอนุกรม พรินเตอร์ หรืออุปกรณ์อื่น ๆ เสมือนหนึ่งว่าเป็นไฟล์ โดยจะครอบคลุมการใช้งานฟังก์ชั่นที่ใช้งานกันบ่อย 5 ฟังก์ชั่นคือ open, close, read, write และ oictl

3.2 ไดเรกทอรี่ที่สำคัญบนยูนิกซ์

สำหรับไดเรกทอรี่เป็นไฟล์ชนิดหนึ่ง การที่จะลบไดเรกทอรี่โดยใช้คำสั่ง rmdir จะต้องไม่มีข้อมูลในไดเรกทอรี่นั้น (ถึงแม้ว่าจะเป็น root ก็ตาม) นอกจากนี้ยังมีคำสั่ง opendir/readdir เพื่อสร้างไดเรกทอรี่

ไดเรกทอรี่เป็นไฟล์เฉพาะรูปแบบหนึ่ง เก็บค่าตารางไอโหนดต่อรายชื่อไฟล์ที่อยู่ในไดเรกทอรี่ รายชื่อของไฟล์ที่อยู่ในไดเรกทอรี่ก็คือ ลิงค์ (link) ไปยังรายชื่อไฟล์บนไอโหนด, การลบไฟล์คือการลบลิงค์ ถ้าไม่มีไฟล์ในไดเรกทอรี่ก็คือ ไม่มีลิงค์ในตารางไอโหนด, พื้นที่ในตารางจะว่าง ไฟล์อื่นสามารถเอาไปใช้งานได้



รูปที่ 3 – 1 ค่าตารางไอโหนดและตำแหน่งที่เก็บบนแผ่นดิสก์

ที่ตารางไอโหนด จะเก็บข้อมูลส่วนอื่นของไฟล์ เช่น วันที่สร้างหรือแก้ไขไฟล์ เพอร์มิสชั่นในการแอ็กเซสไฟล์/ไดเรกทอรี่ ความยาวของไฟล์ หรือตำแหน่งข้อมูลบนดิสก์

ไฟล์จะถูกเก็บไว้ในไดเรกทอรี่ ในรูปแบบโครงสร้างของไฟล์แบบต้นไม้ สำหรับไฟล์ผู้ใช้งานคนนั้น เช่น richie จะมีโฮมไดเรกทอรี่ /home/richie ภายในโฮมไดเรกทอรี่อาจแบ่งเป็นไดเรกทอรี่เก็บ

อีเมล์ จดหมายติดต่อธุรกิจ ทูลที่ใช้ทำงานต่าง ๆ สำหรับโฮมไดเรกทอรี่ ก็เป็นซับไดเรกทอรี่ย่อยในไดเรกทอรี่ที่อยู่เหนือขึ้นไปอีกต่อหนึ่ง, ในกรณีของลีนุกซ์คือ /home

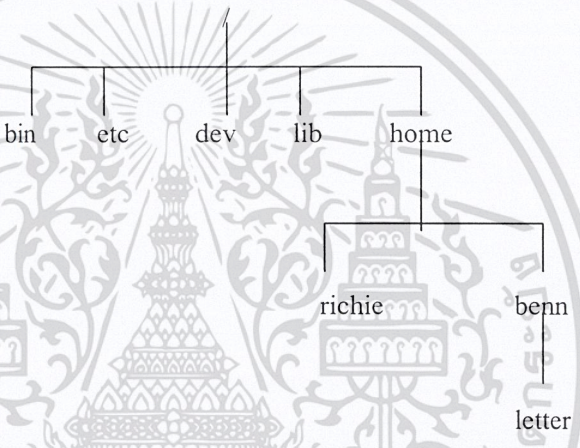
ภายใต้ root, /, นอกจากนี้จะมี /home แล้วยังซับไดเรกทอรี่อื่นอีก เช่น

/bin เก็บโปรแกรมของระบบ

/etc เก็บคอนฟิกกูเลชั่นของระบบ

/lib เก็บไฟล์ไลบรารี่

/dev สำหรับไฟล์ที่ใช้แทน อุปกรณ์รวมทั้งสนับสนุนการติดต่อกับอุปกรณ์เก็บไว้ที่ไดเรกทอรี่



รูปที่ 3 – 2 โครงสร้างของไฟล์แบบต้นไม้

3.3 ไฟล์ของอุปกรณ์

ยูนิกซ์จะมองอุปกรณ์เป็นไฟล์ เช่น ตัวอย่างต่อไปนี้เป็นการ mount ซีดีรอม ให้เป็นไฟล์

```
$ mount –t iso9660 /dev/hdc /mnt/cdrom
$ cd /mnt/cdrom
```

จะสามารถแอ็กเซสซีดีรอมได้โดยผ่านไดเรกทอรี่ /mnt/cdrom สำหรับไฟล์อุปกรณ์ชนิดอื่น ๆ มีดังนี้

/dev/console

เป็นคอนโซลของระบบ ข้อความ error และผลการทำงานต่าง ๆ จะถูกส่งมายังไฟล์นี้ ในยูนิกซ์จะต้องมีเทอร์มินอลที่ได้รับการกำหนดให้เป็นคอนโซลของระบบ อย่างน้อย 1 เทอร์มินอล ส่วนใหญ่จะเป็นคอนโซลที่กำลังแอ็คทีฟ หรือเป็นวินโดวส์ที่กำลังแอ็คทีฟ (ถ้ารันในระบบ X)

/dev/tty

เปรียบเสมือนไฟล์ที่ใช้ติดต่อกับเทอร์มินอลต่าง ๆ ของโปรเซส เช่น คีย์บอร์ด จอภาพเอาต์พุต หรือวินโดวส์ของระบบ X สำหรับ /dev/console มีได้เพียงไฟล์เดียวส่วน /dev/tty มีหลายไฟล์ขึ้นกับอุปกรณ์ที่ใช้งาน

/dev/null

เป็น null device, ข้อมูลที่ถูกเขียนใส่ดีไวซ์ไฟล์นี้ จะถูกละทิ้ง(เหมือนลบข้อมูล นั่นเอง), เมื่อใช้คำสั่ง cp คัดลอกข้อมูลจากไฟล์ /dev/null จะได้ไฟล์เปล่าที่ไม่มีข้อมูลอะไร เช่นการใช้คำสั่งต่อไปนี้

```
$ echo send this message to bin > /dev/null
$ cp  /dev/null empty_file
```

สำหรับอุปกรณ์อื่น ที่พบใน /dev เช่น ฮาร์ดดิสก์หรือฟล็อปปีดิสก์ พอร์ตอนุกรมหรือพอร์ตขนาน เทปไดร์ฟ, ซีดีรอม, การ์ดเสียง หรืออุปกรณ์ภายในของระบบอื่น ๆ จะแสดงสถานะภายในของอุปกรณ์นั้น สำหรับลีนุกซ์ การจะแก้ไขไฟล์พวกนี้ได้จะต้องเป็นซุปเปอร์ยูสเซอร์(superuser) เท่านั้น

ชื่อของดีไวด์ไฟล์ อาจจะแตกต่างกันไปตามระบบปฏิบัติการที่ใช้งาน

3.4 เรียกใช้งานฟังก์ชั่นระดับต่ำ

เราสามารถจะแอ็กเซสไฟล์ หรือควบคุมอุปกรณ์โดยใช้ฟังก์ชั่นระดับต่ำ (System calls) ซึ่งจะมีในระบบยูนิกซ์พร้อมอยู่แล้ว ฟังก์ชั่นเหล่านี้จะได้รับบริการโดยตรงจากส่วนกลางของระบบปฏิบัติการ เรียกว่า เคอร์เนล(Kernel) ซึ่งเป็นกลุ่มของดีไวด์ไดรเวอร์ ประกอบด้วยการอินเทอร์เฟซกับฮาร์ดแวร์ในระดับต่ำ เช่น สำหรับ เทปไดร์ฟ จะประกอบด้วยการเริ่มรันเทปไดร์ฟ, การหมุนไปข้างหน้าหรือกลับหลัง การอ่านหรือการเขียนเทป นอกจากนี้ยังต้องใช้ข้อมูลบางอย่าง เช่น ขนาดของบล็อก รวมทั้งความจำเพาะต่ออุปกรณ์ เช่น เทป เป็นสื่อเก็บข้อมูลแบบ ซีเควนเชี่ยล แอ็กเซส (Sequentail Access) ระบบไม่สามารถจะแอ็กเซสข้อมูลโดยตรงได้

หรือฟังก์ชั่นระดับต่ำของฮาร์ดดิสก์ ก็จะสามารถเขียนข้อมูลที่ตำแหน่งใดของพื้นที่ดิสก์ก็ได้ เนื่องจากเป็นแรนดอมแอ็กเซสดีไวซ์(Random Access device)

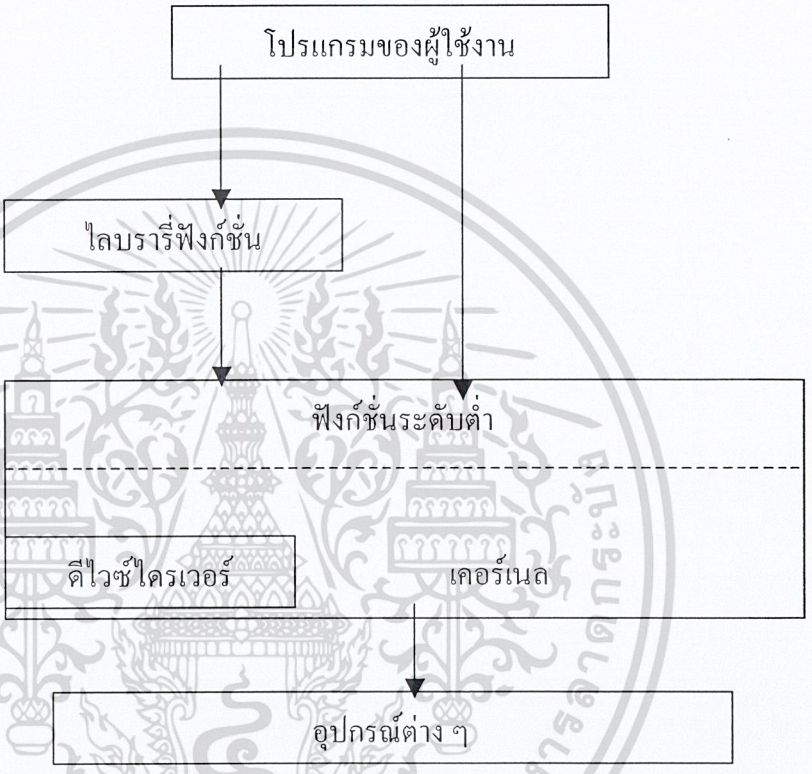
การควบคุมการทำงานเฉพาะอย่างของอุปกรณ์เหล่านี้ ทำได้โดยฟังก์ชั่นระดับต่ำ ioctl ดังนั้นการใช้งานจะแตกต่างกันไปตามแต่ละอุปกรณ์ เช่น การหมุนเทปเพื่อหาตำแหน่งข้อมูลหรือกำหนดโฟลคอนโทรล(flow control) สำหรับพอร์ตอนุกรมเป็นต้น

สำหรับดีไวซ์ไฟล์ใน /dev ส่วนใหญ่แล้ว จะใช้งานฟังก์ชั่นในลักษณะเดียวกันเช่น การแอ็กเซสไฟล์ (open) ไม่ว่าจะเป็น ดิสก์ เทอร์มินอล พรินเตอร์หรือเทปไดร์ฟ

ฟังก์ชั่นระดับต่ำที่เป็นมาตรฐาน ในการแอ็กเซสดีไวซ์ไดรเวอร์(เคอร์เนล) มีดังนี้

open	เปิดไฟล์หรือการเริ่มการติดต่อกับอุปกรณ์
read	อ่านข้อมูลจากไฟล์หรืออุปกรณ์

write	เขียนข้อมูลลงไฟล์หรืออุปกรณ์
close	ปิดไฟล์หรืออุปกรณ์
ioctl	ฟังก์ชั่นควบคุมการทำงานเฉพาะอย่างสำหรับอุปกรณ์



รูปที่ 3 – 3 ฟังก์ชั่นการทำงานระดับต่ำ

3.5 ฟังก์ชั่นการแอ็กเซสไฟล์ในระดับต่ำ

โปรแกรมที่กำลังรันอยู่จะมีโปรเซสการทำงานของตัวเอง โปรเซสเหล่านี้จะทำงานร่วมกับไฟล์หรืออุปกรณ์ โดยใช้หมายเลขแทนไฟล์ (file descriptor) โดยปกติจะมีหมายเลขของ file descriptor อยู่ 3 แบบที่เปิดใช้งานอยู่แล้ว คือ

0	อุปกรณ์อินพุตมาตรฐาน
1	อุปกรณ์เอาต์พุตมาตรฐาน
2	อุปกรณ์แสดงerror

นอกจากนี้ยังจะสามารถทำงานร่วมกับไฟล์หรืออุปกรณ์อื่น ที่มี file descriptor โดยใช้คำสั่ง open เพื่อเริ่มต้น(initialize) การทำงานของ file desciptor และสามารถจะใช้คำสั่ง write หรือ read เพื่อเขียนหรืออ่านข้อมูลจากไฟล์ได้

write

```
#include  <unistd.h>
size_t  write(int fildes, cont void *buf, size_t nbytes);
```

ฟังก์ชั่น write จะนำข้อมูลจำนวน nbytes จากบัฟเฟอร์ buf ไปเขียนลงในไฟล์ที่มี file descriptor fildes โดยจะส่งค่ากลับเป็นจำนวนไบต์ที่เขียนจริง ๆ (ซึ่งอาจจะน้อยกว่า nbytes ก็ได้) ถ้าส่งค่ากลับเป็น 0 แสดงว่าถึงจุดสิ้นสุดของไฟล์ ถ้าส่งค่ากลับเป็น –1 แสดงว่ามี error เกิดขึ้นในการทำงานของ write

read

```
#include  <unistd.h>
size_t  write(int fildes, cont void *buf, size_t nbytes);
```

จะอ่านข้อมูล nbytes จากไฟล์ที่มี file descriptor fildes เก็บไว้ในบัฟเฟอร์ buf จะส่งค่ามาเป็นจำนวนที่อ่านได้จริง ถ้าส่งค่ากลับมาเป็น 0 หมายถึงไม่มีการอ่านข้อมูล ถ้ามี error เกิดขึ้นจะส่งค่ากลับเป็น –1

open

```
#include <fcntl.h>
#include <sys/types.h>
#include  <sys/stat.h>

int open(const char *path, int oflags);
int open(const char *path, int oflags, mode_t mode);
```

> ในมาตรฐาน POSIX (รวมทั้งลีนุกซ์) อาจจะไม่ต้องรวม types.h และ stat.h เข้าไปด้วยแต่ในระบบยูนิกซ์ บางระบบต้องรวมเข้าไปด้วย

คำสั่ง open จะสร้าง path ใหม่ในการแอ็กเซสไฟล์(หรืออุปกรณ์) ถ้าทำได้สำเร็จจะส่งค่ากลับมาเป็นfile descriptor ที่สามรถใช้ได้ในฟังก์ชั่น read, write ได้, file descriptor นี้ จะแตกต่างกันไปแต่ละ โปรเซสไม่สามารถใช้ร่วมกันได้ การทำงานของแต่ละ file descriptor จะต่อเนื่องกันไปเรื่อย เช่น จะอ่านข้อมูลจากจุดที่แล้ว(หน่วยความจำ) หรือจะเขียนข้อมูลต่อจากจุดที่แล้ว หลังจากหยุดทำงาน แต่ละ file descriptor จะมีออฟเซ็ตเป็นของตัวเอง

path จะกำหนดไฟล์ที่เปิดหรือสร้างขึ้นมาใหม่

oflags จะกำหนดเพอร์มิสชั่นการแอ็กเซสเมื่อเปิดไฟล์ โดยจะต้องใช้งานโหมดใดโหมดหนึ่งต่อไปนี้

โหมด	คำอธิบาย
O_RD0NLY	อ่านได้เท่านั้น
O_WR0NLY	สามารถแก้ไขไฟล์ได้
O_RDWR	อ่านและแก้ไขไฟล์ได้

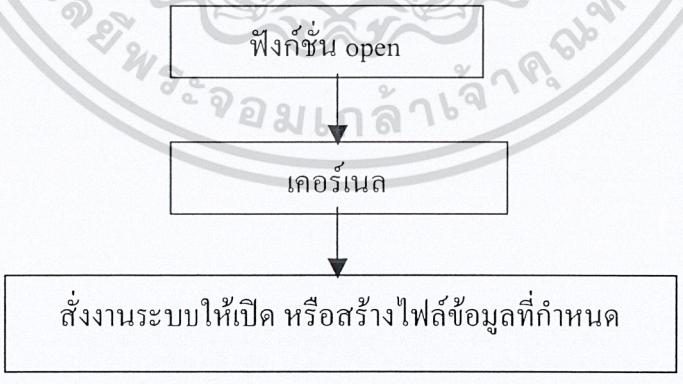
ตาราง 3 – 1 โหมดการทำงาน

นอกจากนี้ยังสามารถใช้งานออฟชั่นข้างต้น OR(bit wise) กับออฟชั่นต่อไปนี้

O_APPEND	เขียนข้อมูลต่อท้ายไฟล์
O_TRUNC	กำหนดความยาวของไฟล์เป็น 0 (ลบข้อมูลถ้ามีอยู่)
O_CREAT	สร้างไฟล์ ถ้าไม่มีไฟล์นั้น โดยใช้เพอร์มิสชั่นในการแอ็กเซสจากข้อมูลในหน้าถัดไป แต่ถ้ามีไฟล์อยู่แล้วจะเปิดไฟล์
O_EXCL	จะใช้ร่วมกับ O_CREAT ในการป้องกันไม่ให้ระบบสร้างไฟล์ดังกล่าวในเวลาเดียวกัน กับโปรแกรมอื่น ในกรณีที่รันในระบบ multitasking ในกรณีมีไฟล์อยู่แล้วจะส่งค่ากลับเป็น –1

ตารางที่ 3 – 2 ออพชั่นเพิ่มเติม

ถ้าทำงานสำเร็จฟังก์ชั่น open จะส่งหมายเลขไฟล์ที่ใช้งานใหม่กลับ(file descriptor) หรือส่ง –1 ถ้าทำงานไม่สำเร็จ และจะสามารถระบุสาเหตุที่ทำงานไม่สำเร็จไว้ที่ตัวแปร errno ค่า file descriptor จะเป็นค่าต่ำสุดที่ไม่มีการใช้งาน



รูปที่ 3 – 4 ขั้นตอนการทำงานของฟังก์ชั่น open

mode เมื่อมีการสร้างไฟล์ใหม่โดยใช้งาน O_CREAT จะต้องกำหนดเพอร์มิสชั่นเริ่มต้นในการแอ็กเซสไฟล์โดยใช้ OR(bits wise) Operator กับแฟลกที่กำหนดไว้ในไฟล์ sys/types.h ดังต่อไปนี้

S_IRUSR กำหนดให้ผู้เป็นเจ้าของไฟล์สามารถอ่านข้อมูลได้

S_IWUSR กำหนดให้ผู้เป็นเจ้าของไฟล์สามารถแก้ไข้ข้อมูลได้

S_IXUSR	กำหนดให้ผู้เป็นเจ้าของไฟล์สามารถจะรันไฟล์ได้เท่านั้น
S_IRGRP	กำหนดให้กลุ่มที่เป็นเจ้าของไฟล์สามารถอ่านข้อมูลได้
S_IWGRP	กำหนดให้กลุ่มที่เป็นเจ้าของไฟล์สามารถแก้ไขข้อมูลได้
S_IXGRP	กำหนดให้กลุ่มที่เป็นเจ้าของไฟล์สามารถจะรันไฟล์ได้
S_IROTH	ผู้ใช้งานคนอื่นนอกเหนือจาก 2 กลุ่มแรกสามารถอ่านข้อมูลได้
S_IWOTH	ผู้ใช้งานคนอื่นนอกเหนือจาก 2 กลุ่มแรกสามารถแก้ไขข้อมูลได้
S_IXOTH	ผู้ใช้งานคนอื่นนอกเหนือจาก 2 กลุ่มแรกสามารถรันไฟล์ข้อมูลได้

นอกจากไฟล์เพอร์มิสชั่นแล้วจะกำหนดโดยคำสั่งข้างต้นแล้ว อีกคำสั่งหนึ่งก็คือ umask ซึ่งเป็นตัวแปรเชลล์ กำหนดระดับการห้ามให้กับเพอร์มิสชั่นของไฟล์เป็นหลักไว้ เมื่อมีการสร้างไฟล์ขึ้นมา

umark

เป็นตัวแปรเชลล์เก็บเพอร์มิสชั่นการแอ็กเซสไฟล์ จะใช้งานเมื่อมีการสร้างไฟล์และสามารถที่จะแก้ไขค่าในตัวแปร umark โดยจะประกอบด้วยตัวเลข 3 หลักในฐาน 8 กำหนดการไม่อนุญาตสำหรับผู้ใช้งาน(user) กลุ่มผู้ใช้งาน (group) หรือผู้ใช้งานอื่น (others) ตามลำดับตามตารางต่อไปนี้

หลัก	ค่า	คำอธิบาย	เพอร์มิสชั่น
1	0	อนุญาตให้เจ้าของมีการแอ็กเซสได้เต็มที่	rwx
	4	ไม่อนุญาตให้เจ้าของอ่านไฟล์ได้	-wx
	2	ไม่อนุญาตให้เจ้าของแก้ไขไฟล์ได้	r-x
	1	ไม่อนุญาตให้เจ้าของรันไฟล์ได้	rw-
2	0	อนุญาตให้กลุ่มผู้ใช้งานมีการแอ็กเซสได้เต็มที่	rwx
	4	ไม่อนุญาตกลุ่มผู้ใช้งานอ่านไฟล์ได้	-wx
	2	ไม่อนุญาตให้ผู้ใช้งานแก้ไขไฟล์ได้	r-x
	1	ไม่อนุญาตให้ผู้ใช้งานรันไฟล์ได้	rw-
3	0	อนุญาตให้ผู้ใช้งานอื่นมีการแอ็กเซสได้เต็มที่	rwx
	4	ไม่อนุญาตให้ผู้ใช้งานอื่นอ่านไฟล์ได้	-wx
	2	ไม่อนุญาตให้ผู้ใช้งานอื่นแก้ไขไฟล์ได้	r-x
	1	ไม่อนุญาตให้ผู้ใช้งานอื่นรันไฟล์ได้	rw-

ตารางที่ 3 – 3 ค่าของ umask

เมื่อมีการสร้างไฟล์ใหม่โดยใช้คำสั่ง open การกำหนดเพอร์มิสชั่นในตัวแปร mode จะถูกเปรียบเทียบกับการห้ามที่กำหนดไว้ในตัวแปร umask , บิตใดที่ถูกยกเลิกใน umask ก็จะถูกยกเลิก(ถ้าต้องการจะ

แก้ไขเพิ่มเพอร์มิสชั่นอื่น จะใช้คำสั่ง chmod แก้ไขภายหลัง) เพื่อป้องกันไม่ให้มีการสร้างไฟล์ที่สามารถแอ็กเซสได้โดยผู้ใช้งานคนอื่นโดยอัตโนมัติ ซึ่งเป็นจุดประสงค์ของ umask

close

```
#include <unistd.h>

int close (int fildes);
```

จะยกเลิกการใช้งานไฟล์เดสคริปเตอร์ กับไฟล์นั้น จะทำให้ฟิลด์สามารถนำไปใช้งานใหม่ได้ จะส่งค่ากลับมาเป็น 0 ถ้าทำงานได้สำเร็จ หรือ ส่งค่ากลับมาเป็น –1 ถ้ามีข้อผิดพลาดเกิดขึ้น

จำนวนไฟล์ที่โปรแกรมที่กำลังรันสามารถจะเปิดได้ในเวลาหนึ่งจะกำหนดโดยตัวแปร OPEN_MAX ใน limits.h แต่อาจจะมีข้อจำกัดอยู่บ้าง เช่น ในระบบ POSIX จะต้องกำหนดไม่ต่ำกว่า 16 เป็นต้น

ioctl

```
#include <unistd.h>

int ioctl (int fildes, int cmd , …);
```

เป็นฟังก์ชั่นสำหรับควบคุมการทำงานเฉพาะอย่างของอุปกรณ์ เช่น เทอร์มินอล, ไฟล์เดสคริปเตอร์, ซอคเก็ต หรือเทปไดร์ฟ ให้ดูรายละเอียดที่ man page ของอุปกรณ์นั้น ๆ

3.6 ฟังก์ชั่นระดับต่ำอื่นในการจัดการไฟล์

นอกจากนี้แล้วภาษาซี ยังมีฟังก์ชั่นระดับต่ำอื่น ๆ ในการจัดการไฟล์เพิ่มเติม ซึ่งมีการใช้งานไม่มากนัก

lseek

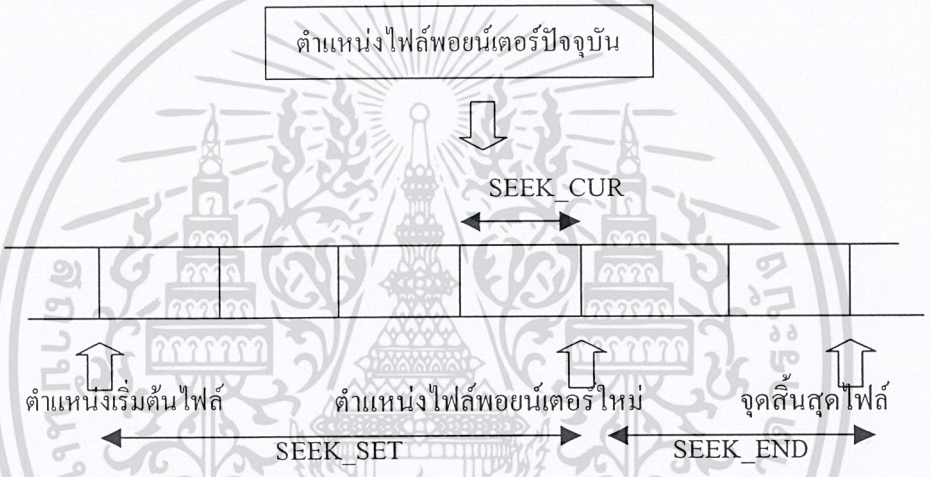
```
#include <unistd.h>
#include <sys/types.h>

off_t     lseek (int fildes,   off_t offset,        int whence);
```

lseek จะกำหนดตำแหน่งพอยน์เตอร์ของไฟล์ที่มีไฟล์เดสคริปเตอร์ฟิลด์ เพื่อกำหนดตำแหน่งที่จะอ่านหรือเขียน สามารถจะกำหนดแบบสัมบูรณ์ หรือ แบบสัมพันธ์กับตำแหน่งปัจจุบันหรือท้ายไฟล์ ค่า offset จะกำหนดตำแหน่งส่วน whence จะกำหนดลักษณะการใช้งานค่าออฟเซ็ต ดังต่อไปนี้

SEEK_SET	offset หมายถึง ค่าสัมบูรณ์จากต้นไฟล์
SEEK_CUR	offset หมายถึงค่าสัมพันธ์กับตำแหน่งปัจจุบัน
SEEK_END	offset หมายถึงค่าสัมพันธ์กับจุดสุดท้ายของไฟล์

คำสั่ง lseek จะส่งค่ากลับเป็นจำนวนไบต์ ออฟเซ็ต วัดจากจุดเริ่มต้นไฟล์ที่พอยน์เตอร์ถูกกำหนด หรือ –1 ถ้ามีข้อผิดพลาดเกิดขึ้นระหว่างการทำงาน, สำหรับตัวแปรชนิด offset_t จะใช้สำหรับออฟเซ็ต ในกรณีที่ต้องการค้นหาข้อมูล แต่บางระบบก็ไม่ใช้งาน



รูปที่ 3 – 5 ตำแหน่งต่างของ lseek

fstat, stat, lstat

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

int     fstat (int fildes,    struct stat *buf);
int     stat (const char *path,     struct stat *buf);
int     lstat (const char *path,    struct stat *buf );
```

fstat	จะส่งสถานะของไฟล์ที่มีหมายเลข fildes ไปเก็บไว้ที่ตำแหน่ง buf
stat และ lstat	จะส่งค่ากลับเช่นเดียวกับ fstat เพียงแต่ถ้าเป็น symbolic link คำสั่ง lstat จะให้เก็บข้อมูลเกี่ยวกับลิงค์ ขณะที่ stat จะให้ข้อมูลไฟล์ที่ลิงค์อ้างถึง

สำหรับโครงสร้างข้อมูล stat มีรายละเอียดดังนี้

ฟิลด์ข้อมูลใน stat	คำอธิบาย
st_mode	ไฟล์เพอร์มิสชั่นและข้อมูลเกี่ยวกับชนิดของไฟล์
st_ino	ข้อมูลไอโหนดที่เกี่ยวกับไฟล์
st_dev	อุปกรณ์ที่เก็บไฟล์นี้
st_uid	Uid เจ้าของไฟล์
st_gid	Gid กลุ่มที่เจ้าของไฟล์
st_atime	เวลาในการแอ็กเซสครั้งล่าสุด
st_ctime	เวลาในการเพอร์มิสชั่นของไฟล์หรือข้อมูลครั้งล่าสุด
st_mtime	เวลาในการแก้ไขข้อมูลของไฟล์ครั้งล่าสุด
st_nlink	จำนวน hard link ของไฟล์

ตารางที่ 3 – 4 โครงสร้างข้อมูล stat

dup และ dup2

```
#include <unistd.h>

int dup (int fildes );
int dup2 (int fildes,        int fildes2);
```

dup จะสร้างไฟล์เดสคริปเตอร์ใหม่ เพื่อให้สามารถแอ็กเซสไฟล์เดียวกันโดยใช้ fil ไฟล์เดสคริปเตอร์ต่างกัน, อาจจะใช้งานในกรณีที่ต้องแอ็กเซส ตำแหน่งของไฟล์ที่ต่างกัน

dup2 จะทำงานเช่นเดียวกันเพียงแต่สามารถจะกำหนดค่าไฟล์เดสคริปเตอร์ใหม่ได้

3.7 ไลบรารี่ฟังก์ชั่น

การใช้งานฟังก์ชั่นระดับต่ำ ค่อนข้างจะขาดความยืดหยุ่น และไม่มีประสิทธิภาพเท่าที่ควร เช่น ด้านข้อจำกัดของฮาร์ดแวร์, เทปไดร์ฟ จะมีขนาดบล็อกอย่างที่ต่ำที่สุด 10 กิโลไบต์ ถ้าต้องการให้เล็กกว่านี้ ฮาร์ดแวร์ก็ยังจะกำหนดให้มีขนาดพื้นที่จริง 10 กิโลไบต์ เหมือนเดิม ทำให้เกิด gap จำนวนมาก

ภาษาระดับสูงที่ใช้อินเตอร์เฟซกับดิสก์หรืออุปกรณ์อื่น ๆ คือ ไลบรารี่ ซึ่งเป็นฟังก์ชั่นมาตรฐานที่สามารถจะรวมเข้าในโปรแกรม เพื่อแก้ปัญหาข้างต้น เช่น เช่น ไลบรารี่เกี่ยวกับการอินพุต/เอาต์พุตข้อมูล ทำให้สามารถจะกำหนดขนาดของบล็อกได้หลายขนาด (ไลบรารี่จะไปเรียกฟังก์ชั่นการทำงานในระดับต่ำอีกต่อหนึ่ง)

3.8 ไลบรารี่ฟังก์ชั่นมาตรฐานสำหรับการอินพุต/เอาต์พุตข้อมูล

จะใช้ไฟล์เฮดเดอร์ stdio.h ติดต่อกับฟังก์ชั่นการทำงานในระดับต่ำ ไลบรารี่จะทำให้การทำงานอินพุต/เอาต์พุตข้อมูลทำให้ง่ายขึ้น นอกจากไลบรารี่ยังทำหน้าที่เกี่ยวกับการกำหนดบัฟเฟอร์สำหรับการทำงานของอุปกรณ์ต่าง ๆ อีกด้วย

การใช้งานจะเหมือนกับการใช้งานฟังก์ชั่นระดับต่ำ จะต้องเปิดไฟล์เสียก่อนเพื่อกำหนดออพเจ็กที่จะใช้อ้างอิงถึงไฟล์ เพียงแต่การเปิดไฟล์จะใช้ไฟล์สตรีมแทนไฟล์เดสคริปเตอร์ กำหนดโดยใช้โครงสร้างพอยน์เตอร์ FILE * เพื่อให้ฟังก์ชั่นอื่นสามารถจะนำไปใช้ในการทำงานกับไฟล์สตรีมได้

ปกติเมื่อมีการรันโปรแกรมจะมีไฟล์สตรีมอยู่ 3 ชนิดที่ถูกเปิดโดยอัตโนมัติคือ stdin, stdout, stderr(จะกำหนดไว้ใน stdio.h) จะเท่ากับค่าไฟล์เดสคริปเตอร์ของฟังก์ชั่นการทำงานในระดับต่ำ 0, 1 และ 2 ตามลำดับ

รูปที่ 3-6 การทำงานของฟังก์ชั่นระดับต่ำ

fopen

```
#include <stdio.h>
FILE    *fopen(const char *filename,        const char  *mode);
```

เป็นฟังก์ชั่นที่สร้างขึ้นมาจาก open ซึ่งเป็นฟังชั่นการทำงานระดับต่ำอีกต่อหนึ่ง ส่วนใหญ่จะใช้ในการอินพุต/เอาต์พุตข้อมูล แต่ถ้าต้องการจะแอ็กเซสไฟล์(อุปกรณ์) โดยตรง ขอแนะนำให้ใช้ open เนื่องจากไม่มีผลข้างเคียงจากการทำงานของไลบรารี่ เช่น การกำหนดบัฟเฟอร์ เป็นต้น

ฟังก์ชั่นจะเปิดไฟล์ชื่อ filename โดยกำหนดลักษณะการเปิดไฟล์ในพารามิเตอร์ mode ดังต่อไปนี้

“r” หรือ “rb”	เปิดไฟล์เก่าเพื่ออ่านอย่างเดียว
“w” หรือ “wb”	เปิดไฟล์ใหม่เพื่อเขียนอย่างเดียวถ้ามีไฟล์เก่าจะเขียนทับ
“a” หรือ “ab”	เปิดไฟล์เก่าเพื่อเขียนเพิ่มที่ท้ายไฟล์
“r+” หรือ “rb+” หรือ “r+b”	เปิดไฟล์เก่าเพื่ออัพเดทข้อมูล
“w+” หรือ “wb+”หรือ “w+b”	เปิดไฟล์ใหม่สำหรับอ่านและเขียน ถ้ามีไฟล์เก่าจะเขียนทับ
“a+” หรือ “ab+”หรือ “a+b”	เปิดไฟล์เก่าเพื่ออ่าและเขียนข้อมูลที่ท้ายไฟล์ ถ้าไฟล์เก่าไม่มีจะเปิดไฟล์ใหม่เพื่ออ่านและเขียน

สำหรับอักษร b จะระบุเป็นไฟล์ไบนารี่ อันที่จริงแล้วยูนิกซ์จะทำงานกับไฟล์เหมือนกันไม่ว่าจะเป็นเท็กซ์ไฟล์ หรือ ไบนารีไฟล์ คือจะทำงานเสมือนว่าเป็นไบนารี่ไฟล์ทั้งหมด เช่น การแปลอักขระควบคุมบรรทัด, สำหรับการกำหนดโหมดนั้น จะต้องกำหนดให้เป็นข้อความคือ ใช้ “r” ไม่ใช่ตัวอักษร ‘r’

ถ้าฟังก์ชั่นทำงานสำเร็จจะส่งค่ากลับมาเป็นพอยน์เตอร์ชี้ไปยังไฟล์นั้น ถ้าทำงานไม่สำเร็จจะส่งค่ากลับมาเป็น NULL (กำหนดไว้ใน stdio.h)

fclose

```
#include <stdio.h>
int     fclose (FILE *stream);
```

จะปิดไฟล์สตรีม stream , การใช้คำสั่ง fclose เป็นสิ่งจำเป็นเนื่องจากไลบรารี่ stdio.h จะเก็บข้อมูลไว้ในบัฟเฟอร์ เมื่อใช้คำสั่งนี้จะทำให้โปรแกรมเขียนข้อมูลในบัฟเฟอร์กลับไปยังที่ไฟล์ อย่างไรก็ตามไฟล์จะถูกปิดโดยอัตโนมัติเมื่อสิ้นสุดการทำงานของโปรแกรม

จำนวนของไฟล์ที่สามารถเปิดได้สูงสุด จะถูกกำหนดไว้ที่ตัวแปร FOPEN_MAX ใน stdio.h

fread

```
#include <stdio.h>
size_t  fread(void *ptr,  size_t size,      size_t  nitem,      FILE stream);
```

จะอ่านข้อมูลจากไฟล์สตรีม stream ไปเก็บไว้ในบัฟเฟอร์ ptr ทั้งฟังก์ชั่น fread, fwrite จะจัดการข้อมูลเป็นเร็กคอร์ด ขนาดของเร็กคอร์ดจะกำหนดโดย size จำนวนเร็กคอร์ดที่อ่านจะกำหนดโดย nitem หลังจากอ่านข้อมูลเสร็จจะส่งจำนวนเร็กคอร์ดที่อ่านได้กลับ อนึ่งฟังก์ชั่นจะทำหน้าที่อ่านและเก็บข้อมูลไปยังบัฟเฟอร์เท่านั้น เป็นหน้าที่ของโปรแกรมเมอร์ที่จะแยกแยะข้อมูลเหล่านั้นด้วยตัวเอง

fwrite

```
#include <stdio.h>
size_t  fread(const void *ptr,  size_t size,  size_t  nitem,  FILE
        stream);
```

จะทำหน้าที่ตรงกันข้ามกับฟังก์ fread โดยจะนำข้อมูลจากบัฟเฟอร์ ptr เขียนลงไปที่ไฟล์สตรีม stream และส่งค่ากลับเป็นเร็กคอร์ดที่เขียนได้ ในกรณีเป็นจอมอนิเตอร์ stream จะเป็น stdout

fflush

```
#include <stdio.h>
int fflush (FILE *stream);
```

จะเขียนข้อมูลของไฟล์สตรีม stream ไปเก็บที่ดิสก์ทันที จะใช้ในการเก็บข้อมูลลงดิสก์เพื่อป้องกันการสูญหาย ในระหว่างการรันโปรแกรม เมื่อใช้ fclose จะเรียกฟังก์ชั่น fflush โดยอัตโนมัติ

fseek

```
#include <stdio.h>
int   fseek (FILE *stream,  long int offset,    int whence);
```

จะกำหนดตำแหน่งบนไฟล์ สำหรับการอ่านหรือเขียน offset และ whence จะใช้งานเหมือนฟังก์ชั่นระดับต่ำ lseek อย่างไรก็ตาม lseek จะส่งค่า offset จากต้นไฟล์กลับ แต่ fseek จะส่ง 0 ถ้าทำงานสำเร็จ และ –1 ถ้าทำงานไม่สำเร็จ ขณะที่ตัวแปร errno จะระบุข้อผิดพลาดที่เกิดขึ้น

fgetc, getc, getchar

```
#include <stdio.h>
int fget (FILE *stream);
int gets (FILE *stream);
int  getchar ();
```

fgetc จะอ่านข้อมูล 1 ไบต์จาก stream และเลื่อนพอยน์เตอร์ชี้ไปยังไบต์ถัดไป เมื่อถึงจุดสิ้นสุดของไฟล์จะส่งค่า EOF กลับ

getc จะทำงานคล้ายกลับ fgetc เพียงแต่ทำงานในลักษณะของมาโคร

getchar จะเท่ากับ getc(stdin) สำหรับอ่านข้อมูลหนึ่งตัวอักษรจากอุปกรณ์อินพุตมาตรฐาน

fputc, putc, putchar

```
#include <stdio.h>
int fputc (int c,    FILE *stream);
int putc  (int c,    FILE *stream);
int  putchar (int c);
```

fputc จะเขียนข้อมูลไปยังเอาต์พุตสตรีม เมื่อถึงจุดสิ้นสุดไฟล์จะส่ง EOF กลับ

putc จะคล้ายกลับ fputc เพียงแต่ทำงานในลักษณะของมาโคร

putchar จะเท่ากลับ putc (c, stdout) เขียนข้อมูลหนึ่งตัวอักษร ไปยังอุปกรณ์เอาต์พุตมาตรฐาน เมื่อสินสุดไฟล์จะส่งค่า –1 กลับ (ไม่ใช่ EOF)

fgets, gets

```
#include <stdio.h>
char *fgets(char *s,        int n,    FILE *stream);
char  *gets(char *s);
```

fgets จะอ่านข้อมูลจากอินพุตสตรีม แล้วเขียนข้อมูลลงไปที่ตำแหน่งพอยต์เตอร์ s จนกระทั่งพบตัวอักษร newline หรือ รับข้อมูลไปแล้วเป็นจำนวน n - 1 ตัวอักษรหรือสิ้นสุดไฟล์ หรืออ่านข้อมูลเสร็จจะเพิ่มตัวอักษร null byte, \0 , เข้าไปที่ท้ายข้อความ รวมเป็นตัวอักษรทั้งหมด n ไบต์

เมื่อทำงานเสร็จ fgets จะส่งค่าเป็นพอยน์เตอร์ชี้ไปยังข้อความ s ถ้าข้อมูลอินพุตสตรีมจะอยู่ที่จุดสิ้นสุดไฟล์ จะส่งค่ากลับเป็น null pointer ถ้ามีข้อผิดพลาดเกิดขึ้นระหว่างการอ่าน จะระบุข้อผิดพลาดที่เกิดขึ้นไว้ในตัวแปร errno

gets จะอ่านข้อมูลจากอินพุตมาตรฐาน และตัดตัวอักษร newline ทิ้ง และเพิ่มตัวอักษร null byte เช่นเดียวกับ fgets และอีกอย่างหนึ่งที่น่าสนใจก็คือ gets ไม่มีการจำกัดจำนวนตัวอักษรที่จะอ่าน ดังนั้นข้อมูลที่อยู่ในบัฟเฟอร์อาจจะถูกเขียนทับได้

3.9 ฟังก์ชั่นจัดการรูปแบบการอินพุตและเอาต์พุตข้อมูล

มีฟังก์ชั่นอยู่จำนวนหนึ่ง สำหรับควบคุมการแสดงข้อมูลให้เป็นไปตามที่ต้องการเช่น printf หรือ scanf สำหรับอ่านข้อมูลในรูปแบบที่ต้องการ

printf, fprintf, sprintf

```
#include <stdio.h>
int     printf (const char *format, …);
int     fprintf (FILE *stream,     const char *format, …);
int     sprintf(char *s,    const char *format);
```

printf จะแสดงข้อมูลทางอุปกรณ์เอาต์พุตมาตรฐาน (กำหนดโดยพารามิเตอร์ format)

fprintf เขียนข้อมูลไปยังสตรีมที่ต้องการ

sprintf จะเขียนข้อมูลไปยังตัวแปรสตริง s ดังนั้นตัวแปรนี้จะต้องมีความจุมากพอสำหรับข้อมูล สำหรับข้อมูลทั่วไปจะเขียนลงไปตามปกติ ยกเว้น คอนเวอร์ชั่นสเปคซิฟายเออร์(convsersion specifier) เพื่อกำหนดรูปแบบการแสดงข้อมูล (ส่วนใหญ่จะเริ่มด้วยตัวอักษร %)

scanf, fscanf, sscanf

```
#include < stdio.h >
int scanf (const char *format,…);
int  fscanf (FILE *stream,  const char *format, …);
int  sscanf (const char *s,   const char *format);
```

ฟังก์ชั่นในตระกูล scanf จะทำงานคล้ายกับฟังก์ชั่นตระกูล printf เพียงแต่จะอ่านข้อมูลจากอินพุตสตรีม เก็บไว้ยังตำแหน่งหน่วยความจำที่กำหนด สำหรับรูปแบบของข้อมูลที่อ่านจะกำหนดโดย คอนเวอร์ชั่นสเปคซิฟายเออร์ ส่วนใหญ่จะคล้ายกับที่ใช้งานในฟังก์ชั่นตระกูล printf

สำหรับรูปแบบของข้อมูล scanf จะประกอบด้วยตัวอักษรทั่วไปและ คอนเวอร์ชั่นสเปคซิฟายเออร์ แต่ต่างกันที่ คอนเวอร์ชั่นสเปคซิฟายเออร์ จะวางไว้ที่สวนอินพุตของข้อมูล

ฟังก์ชั่น scanf จะส่งค่ากลับเป็นจำนวนเร็กคอร์ดที่อ่านได้ ถ้ามีการอ่านจนถึงจุดสิ้นสุดไฟล์จะส่งค่า EOF กลับ ถ้ามี error เกิดขึ้นในระหว่างการอ่านข้อมูลจากอินพุตสตรีม จะกำหนด error flag เพื่อแจ้งการเกิดความผิดพลาด พร้อมกับระบุชนิดของความผิดพลาดไว้ที่ตัวแปร errno

3.10 การจัดการกับไฟล์และไดเร็กทอรี

ไลบรารี่ฟังก์ชั่นสำหรับทำงานกับไฟล์และไดเร็กทอรี่ จะมีดังนี้

chmod

จะเปลี่ยนเพอร์มิสชั่นในการแอ็กเซสไฟล์ หรือ ไดเร็กทอรี่

```
#include < sys/stat.h>
int     chmod (const char *path,   mode_t  mode);
```

ไฟล์ที่ต้องการเปลี่ยนเพอร์มิสชันจะกำหนดโดย path โดยใช้เพอร์มิสชั่น mode เช่นเดียวกับ open

unlink, link, symlink

```
#include < unistd.h>

int     unlink (const char * path);
int     link (const char *path1,     const char *path2);
int     symlink(const char *path1,const char *path2);
```

unlink ใช้สำหรับลบลิงค์ไฟล์ที่ต้องการ จะส่งค่ากลับมาเป็น 0 ถ้าทำงานสำเร็จ หรือ –1 ถ้ามีข้อผิดพลาดเกิดขึ้น อย่างไรก็ตามการจะลบได้ต้องมีเพอร์มิสชั่นที่เพียงพอด้วย โปรแกรม rm ก็ใช้ฟังก์การทำงานนี้

link จะสร้างลิงค์ของไฟล์ path1 ชื่อ path2 สำหรับการสร้างซิมโบริค ลิงค์จะใช้ฟังก์ชั่น symlink ใช้งานในทำนองเดียวกัน ในเชลล์จะใช้คำสั่ง ln สร้างลิงค์ที่ต้องการ ในขณะที่การเขียนโปรแกรมจะใช้ฟังก์ชั่น link แทน

> ลิงค์ (link)จะมีอยู่ 2 ชนิด คือฮาร์ดลิงค์(hard link) เป็นชื่อไฟล์ในตารางไอโหนด ที่ชี้ไปยังตำแหน่งข้อมูลเดียวกันบนฮาร์ดดิสก์ และซิมบริค ลิงค์(symblic link) เป็นเท็กซ์ไฟล์เก็บข้อความระบุตำแหน่งของไฟล์(ไม่อยู่ในตารางไอโหนด)

mkdir, rmdir

```
#include <sys/type.h>
int     mkdir (const char *path,    mode_t  mode);
```

ฟังก์ชั่น mkdir จะสร้างไดเร็กทอรี่ path ที่กำหนด โดยใช้เพอร์มิสชั่น mode (สามารถใช้งานออพชั่น O_CREAT ของฟังก์ชั่น open และได้รับผลจากคำสั่ง umask ด้วยเช่นกัน)

```
#include < unistd.h>
int         rmdir (const char *path);
```

ฟังก์ชั่น rmdir จะลบไดเร็กทอรี่ path ที่ไม่มีข้อมูลในไดเร็กทอรี่ ออกจากระบบ

chdir, getwd

ฟังก์ชั่นระดับต่ำ chdir จะเปลี่ยนไปยังไดเร็กทอรี่ที่ต้องการ(เหมือนกับคำสั่ง cd บนเชลล์)

```
#include <unistd.h >
int         chdir(const char *path);
```

ส่วนไลบรารี่ฟังก์ชั่น getwd จะส่งค่ากลับเป็นไดเร็กทอรี่ที่กำลังทำงานอยู่ปัจจุบัน

```
#include <unistd.h>
char        *getwd(char  *buf,          size_t  size);
```

โดยจะเขียนชื่อของไดเร็กทอรี่ปัจจุบันในบัฟเฟอร์ buf หรือจะส่งค่ากลับมาเป็น null ถ้ามี error เกิดขึ้น เช่น ชื่อมีความยาวมากกว่าความจุบัฟเฟอร์ ซึ่งกำหนดโดยพารามิเตอร์ size

3.11 ตรวจสอบไดเร็กทอรี่

ไลบรารี่ฟังก์ชั่นได้ถูกพัฒนาขึ้นมาเพื่อให้ทำงานกับไดเร็กทอรี่ทำได้ง่ายขึ้น ฟังก์ชั่นกำหนดไว้ที่ไฟล์ dirent.h โดยชื่อโครงสร้างข้อมูลชื่อ DIR สำหรับจัดการกับไดเร็กทอรี่ พอยน์เตอร์ชี้ไปยังโครงสร้างนี้เรียกว่า directory stream pointer (DIR) ฟังก์ชั่นที่ทำงานกับไดเรกทอรี่จะมีอยู่หลายอย่างดังต่อไปนี้

opendir

```
#include <sys/types.h>
#include <dirent.h>
DIR         *opendir (const char *name);
```

ฟังก์ชั่น opendir จะเปิด(เริ่ม)การทำงานกับไดเร็กทอรี่ ถ้าทำสำเร็จจะส่งค่ากลับเป็นพอยน์เตอร์ชี้ไปยังโครงสร้างข้อมูล DIR เพื่ออ่านข้อมูลในไดเร็กทอรี่ ถ้าทำงานไม่สำเร็จจะส่งค่ากลับมาเป็น null pointer

readdir

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

```
struct   dirent   *readdir(DIR      *dirp);
```

จะส่งค่ากลับเป็นพอยน์เตอร์ชี้ไปยังข้อมูลถัดไปในไดเร็กทอรี่สตรีม dirp เมื่อถึงจุดสิ้นสุดข้อมูล จะส่งค่ากลับเป็น null สำหรับโครงสร้างข้อมูล dirent จะเก็บข้อมูลของไฟล์ในไดเร็กทอรี่ที่สำคัญคือ

ino_t	d_ino	inode ของไฟล์
char	d_name[]	ชื่อไฟล์

telldir

```
#include <sys/types.h>
#include <dirent.h>
long int  teedir (DIR  *dirp);
```

จะส่งค่ากลับเป็นจำนวนตัวเลขแสดงตำแหน่งปัจจุบันในไดเร็กทอรี่สตรีม จะใช้งานก่อนที่จะใช้ seekdir

seekdir

```
#include <sys/types.h>
#include <dirent.h>
void      seekdir(DIR  *dirp,        long int  loc);
```

จะกำหนดตำแหน่งของพอยน์เตอร์ในไดเร็กทอรี่สตรีม dirp ค่า loc จะกำหนดตำแหน่งที่ต้องการ ถ้าต้องการรีเซ็ตพอยน์เตอร์ให้อยู่ในตำแหน่งปัจจุบัน ควรใช้ค่าส่งกลับจากฟังก์ชั่น telldir

closedir

```
#include <sys/types.h>
#include <dirent.h>
int       closedir (DIR *dirp);
```

จะปิดการทำงานของไดเร็กทอรี่สตรีม รีซอร์สต่าง ๆ ที่ใช้สำหรับไดเร็กทอรี่สตรีม สามารถจะนำไปใช้กลับงานอื่น ๆ ได้ จะส่งค่ากลับเป็น 0 ถ้าทำงานสำเร็จ หรือ –1 ถ้าทำงานไม่สำเร็จ

3.12 การระบุ error ของฟังก์ชั่น

หลายฟังก์ชั่นในไลบรารี่ stdio จะส่งค่าความผิดพลาดกลับเป็นค่าที่เป็นไปไม่ได้ เช่น จำนวนที่มากจนเกินไป หรือ null pointer หรือ EOF ส่วนการระบุชนิดของความผิดพลาดที่เกิดขึ้นจะเก็บไว้ที่ตัวแปร errno ซึ่งเป็นตัวแปรที่ใช้งานกันได้ทั่วโปรแกรม (External variable)

```
#include <errno.h>
extern   int      errno;
```

นอกจากนี้ยังยังมีฟังก์ชั่นที่ใช้ค้นหาว่า มีข้อผิดพลาดเกิดขึ้นหรือไม่ หรือ ถึงจุดสิ้นสุดไฟล์หรือยัง โดยใช้ฟังก์ชั่นการทำงานต่อไปนี้

```
#include <stdio.h>
int     ferror(FILE  *stream);
int     feof (FILE  *stream);
void    clearerr (FILE  *stream);
```

ferror จะทดสอบว่ามีการกำหนดค่า errno หรือเปล่า ส่งค่ากลับเป็นตัวเลขไม่เท่ากับ 0 ถ้ามีการกำหนด หรือจะส่งค่ากลับเป็น 0 ถ้าไม่มีการกำหนดค่าให้กับ errno

feof จะทดสอบว่าถึงจุดสิ้นสุดไฟล์หรือยัง ถ้ายังไม่ถึงจุดสิ้นสุดไฟล์จะส่งค่าที่ไม่เท่ากับ 0 กลับแต่ถ้าถึงจุดสิ้นสุดไฟล์จะส่งค่า 0

clearerr จะลบข้อมูล EOF หรือข้อมูลใน errno ที่stream ชี้อยู่ ไม่มีการส่งค่ากลับเป็นการเคลียร์ตัวแปรเท่านั้น

การใช้งานตัวแปร errno เป็นวิธีการทั่ว ๆ ไปสำหรับไลบรารี่ ดังนั้นถ้าเป็นการทำงานที่อาจจะมีข้อผิดพลาดเกิดขึ้น ควรจะตรวจสอบตัวแปรนี้ทันที เนื่องจากอาจถูกเขียนทับโดยฟังก์ชั่นการทำงานอื่น ค่าที่กำหนดให้กับค่าของตัวแปร errno จะกำหนดไว้ในไฟล์ errno.h ดังต่อไปนี้

EBUSY	อุปกรณ์หรือ รีซอร์สไม่สามารถทำงานตามที่ต้องการ
EEXIST	มีไฟล์อยู่
ENODEV	ไม่มีอุปกรณ์
EISDIR	เป็นไดเร็กทอรี่
ENOTDIR	ไม่ใช่ไดเร็กทอรี่
EINVAL	ใช้งานอาร์กิวเมนต์ไม่ถูกต้อง
EMFILE	เปิดไฟล์มากเกินไป
EINTR	เกิดการอินเตอร์รัพในระหว่างการทำงานของฟังก์ชั่นระดับต่ำ
EIO	เกิดข้อผิดพลาดในกระบวนการ อินพุต/เอาต์พุต
EPERM	ไม่ได้รับเพอร์มิสชั่นสำหรับการทำงานดังกล่าว
ENOENT	ไม่มีไฟล์หรือไดเร็กทอรี่

มีฟังก์ชั่นบางอย่างในการทำงานกับความผิดพลาดคือ strerror และ perror

```
#include <string.h>
char      *strerror (int  errnum);
```

ฟังก์ชั่น strerror จะแม็พหมายเลขของความผิดพลาด เป็นคำอธิบายที่เข้าใจได้ง่าย ใช้สำหรับบันทึกใน ไฟล์ log

```
#include <stdio.h>
void      perror (const  char  *s);
```

ฟังก์ชั่น perror จะแม็พความผิดพลาดเป็นคำอธิบายและแสดงออกทาง standard error stream และสามารถกำหนดให้นำหน้าตัวอักษรในสตริง s ตามด้วยเครื่องหมายโคลอนและช่องว่าง(ไม่มีก็ได้, null)

บทที่ 4
การตรวจสอบความถูกต้องของข้อมูล
โดยการคำนวณแบบ MD5

4.1 เนื้อหาโดยสังเขป

ในส่วนนี้จะเป็นการอธิบายอัลกอริทึมของ MD5 เมสเซจไดเจสซึ่งจะเป็นเอาอินพุตที่มีความยาวที่กำหนดเอาเอง และมาสร้างเอาต์พุตที่มีขนาด 128 บิต จาก"fingerprint" หรือ "message digest" เป็นการคาดเดาเพื่อที่จะคำนวณที่จะสร้างข้อความ 2 ข้อความ ที่มีความสำคัญของข้อความที่คล้ายกัน หรือการสร้างข้อความที่มีจุดมุ่งหมายทีกำหนดให้เป็นไปไม่ได้ MD5จะถูกนำไปไปใช้การตรวจสอบลายมือชื่อดิจิตอล(Digital Signature) เมื่อแฟ้มข้อมูลมีขนาดใหญ่จะต้องมีการบีบอัดข้อมูลแบบปลอดภัยก่อนที่จะมีการเข้ารหัสด้วยคีย์หลัก(คีย์ที่เป็นความลับ) ภายใต้คีย์สาธารณะของระบบเข้ารหัส(Cryptosystem) อย่างเช่น อาร์เอสเอ(RSA)

MD5 ได้ออกแบบให้ใช้งานด้วยความเร็วบนเครื่อง 32 บิต แต่ถึงอย่างไรอัลกอริทึมของ MD5 ก็ไม่ต้องการตารางที่ใช้แทนจำนวนมาก(อัลกอริทึมสามารถที่จะมีโค๊ดได้อย่างเร็วและกระชับ)

อัลกอริทึมของMD5 เป็นส่วนที่เพิ่มเติมมาจากอัลกอริทึมของMD4 เมสเซจไดเจส ซึ่งการทำงานของ MD5 จะช้ากว่าการทำงานของ MD4 แต่MD5 จะออกแบบให้ครอบคลุมกว่า MD4 ซึ่งMD5 ถูกสร้างขึ้นเพราะว่ามีผู้สงสัยว่า MD4 จะใช้งานได้เร็วแต่มีขีดจำกัดโดยมีผู้คนมาวิจารณ์ว่ามีข้อบกพร่องเมื่อมีการใช้งาน และMD4 ถูกออกแบบให้ทำงานได้เร็วในการทำงานปกติซึ่งเสี่ยงต่อการที่จะถูกโจมตี ซึ่งMD5 ได้ออกแบบให้ทำงานได้ช้ากว่าเล็กน้อยแต่มีความปลอดภัยเป็นพิเศษ โดยเป็นการรวมเอาข้อวิจารณ์ของ MD4 มาทำการแก้ไขแล้วมาสร้างMD5 ขึ้นซึ่งMD5 จะสามารถทำงานได้บนโดเมนมาตรฐานซึ่งตัวระบุมาตรฐานของ MD5 คือ

MD5 OBJECT IDENTIFIER ::=

iso(1) member-body(2) US(840)rsadsi(113549)digestAlgorithm(2)5

4.2 คำศัพท์และความหมายของสิ่งที่กล่าวในบทนี้

ในบทนี้ เวิร์ด(Word) มีขนาด 32 บิต และ ไบต์(Byte) จะมีขนาด 16 บิต ลำดับของบิต(Bit) จะมีลักษณะแบบธรรมชาติเหมือนกับลำดับของไบต์ เมื่อการติดต่อของแต่ละกลุ่มคือ 8 บิต จะหมายถึงไบต์ด้วย high – order

ให้ x_i คือ x ลบด้วย i ถ้าส่วนที่ขีดเส้นใต้เป็นสมการเราจะตัดสินใจล้อมรอบ เช่น x_[i+1] ในทางเดียวกันเราจะใช้ ^ สำหรับการเขียนไว้ข้างบน(เอ็กซ์โพเนนเชี่ยน) ดังนั้น x ^ i นั้นหมายถึง x ยกกำลัง

ด้วย i , สัญลักษณ์ “+” แทนการบวกเพิ่มของเวิร์ด(เช่น การเพิ่มมอดูโล –2^32) , X <<< s การเกิดค่า 32 บิตโดยการเลื่อนบิตของ X เป็นวงกลม(การหมุน) ไปทางซ้ายจำวน s ตำแหน่ง, not(X) เป็นการกลับบิตเป็นคอมพลีเมนต์(Complement) ของ X และให้ X v Y เป็นการออร์ (OR)ของบิตของ X กับ Y , X xor Y เป็นการเอ็กซ์คูซีพ-ออร์(xor) ของ X กับ Y และ XY เป็นการแอนด์(AND) บิตของ X และ Y

4.3 MD5 อัลกอริทึม (MD5 Algorithm)

เราจะเริ่มโดยการมีข้อความ b เป็นอินพุต(Input) เราต้องการหาแมสเซสดิเจส(Message Digest) ข้อความ b จะไม่เป็นจำนวนเต็มที่เป็นลบ b จะต้องเป็น 0 ไม่ต้องการผลคูณด้วย 8 และไม่ต้องการเกณฑ์ขนาดใหญ่เราลองจินตนาการบิตของข้อความที่เขียนลงจะเป็นดังนี้

m_0 m_1……..m[b-1]

ปฏิบัติตาม 5 ขั้นตอนดังต่อไปนี้ในการคำนวณแมสเซสดิเจสของข้อความ

ขั้นตอนที่ 1 การเพิ่มบิตแพดดิ้ง (Append Padding Bits)

ข้อความเป็นแพด(ส่วนขยาย) ซึ่งมีความยาว(เป็นบิต)ที่มอดูโล 512 แล้วเท่ากับ 448 หากไม่ถึงหรือเกินให้เพิ่มเติมแพดดิ้งเข้าไป ดังนั้นข้อความที่จะขยายจะต้องมีอย่างน้อย64บิตของผลคูณของ 512 จะทำการแพดดิงเสมอ ถ้าความยาวของข้อความจริงหารด้วย 512 เท่ากับ 448 การทำแพดดิ้งจะทำแบบนี้ เป็นบิต 1 เดียว เป็นตัวเพิ่มเติมของข้อความและบิตเป็น 0 เป็นการเพิ่มเติมข้อความในบิตแพดดิ้งให้หารด้วย 512 มีค่าเท่ากับ 448 ดังนั้นบิตที่ต้องขยายน้อยสุด 1 บิตและขยายมากสุด 512 บิต

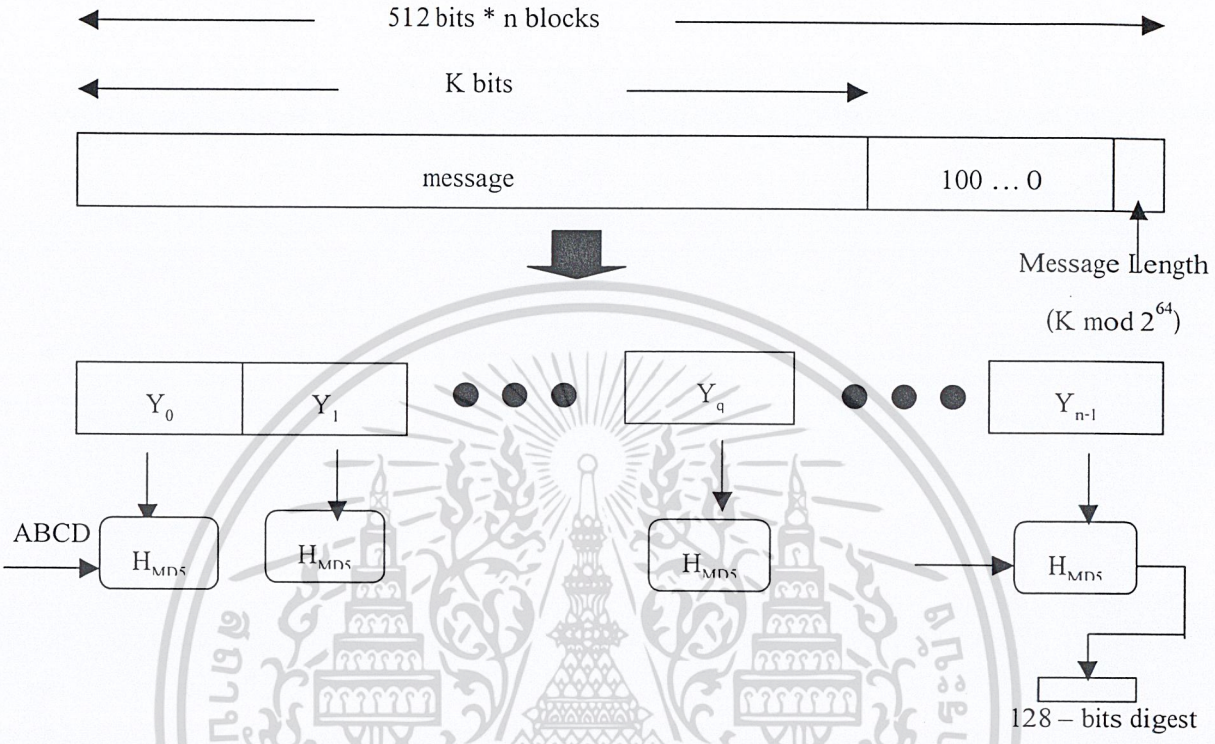
ขั้นตอนที่ 2 เพิ่มเติมความยาว (Append Lengh)

64 บิตของ b (ความยาวของข้อความก่อนการเพิ่มบิตแพดดิ้ง) เป็นการขยายขั้นตอนก่อนหน้า ดังนั้น b ต้องไม่น่ามีค่ามากกว่า 2 ^ 64 เมื่อมีแค่ 64 ตำเท่านั้นที่ใช้(จะมีการขยาย 32 บิตเวิร์ด และขยายบิตเวิร์ดต่ำที่ตรงกับค่าก่อนหน้านี้) ที่จุดนี้เป็นผลของข้อความ(ด้วยการรวมบิตแพดกับ b) ความยาวที่แน่นอนจะเป็นของผลคูณ 16(32 บิต)เวิร์ด โดย M[0 … N-1] แทนผลของข้อความ เมื่อ N เป็นผลคูณของ 16

ขั้นตอนที่ 3 การกำหนดค่าเริ่มต้นแมสเซสดิเจสบัพเฟอร์ (MD Buffer)

บัพเฟอร์จะมี 4 ส่วน(A, B, C, D) ที่ใช้ในการคำนวณแมสเซสดิเจสแต่ละตัวของ A, B, C, D เป็นรีจีสเตอร์ 32 บิต โดยรีจีสเตอร์เหล่านี้จะมีค่าเริ่มต้นจะมีค่าเริ่มต้นในฐานสิบหก

เวิร์ด A	: 01	23	45	67
เวิร์ด B	: 89	ab	cd	ef
เวิร์ด C	: ef	dc	ba	98
เวิร์ด D	: 76	54	32	10



รูปที่ 4 – 1 แสดงการคำนวณแบบ MD5

ขั้นตอนที่ 4 กระทำข้อความใน 16 เวิร์ดบล็อก (Process Message in 16 – Word Block)

เริ่มด้วยการกำหนด 4 ฟังส์ชั่นช่วยโดยเอา 32 บิตเวิร์ด 3ตัว และมาสร้างเอาท์พุต 32 บิตเวิร์ด 1 ตัว

F(X, Y, Z) = XY v not(X)Z

G(X, Y, Z) = XZ v Ynot(Z)

H(X, Y, Z) = X xor Y xor Z

I(X, Y, Z) = Y xor (X v not(Z))

ในแต่ละตำแหน่งบิตที่กระทำ F มีเงื่อนไขดังนี้ : ถ้า X เป็นจริง จะได้ Y ถ้าเป็นเท็จจะได้ Z ฟังก์ชั่น F จะกำหนดการใช้ + การแทนที่ของ v(or) เพราะ XY และ not(X)Z จะไม่เป็น 1 ในตำแหน่งบิตเดียวกัน ที่สนใจก็คือ ถ้าบิตของ X, Y, และ Z เป็นอิสระต่อกันและตรงไปตรงมา ดังนั้นแต่ละบิตของฟังก์ชั่น F(X, Y, Z) จะเป็นอิสระต่อกัน

ฟังก์ชั่น G, H, และ I จะมีลักษณะการทำงานคล้ายกับฟังก์ชั่น F ซึ่งการทำงานจะทำแบบบิตไวส์ พาราแลล (Bitwise parallel) ที่จะสร้างเอาท์พุตจากบิตของ X, Y, และ Z แบบที่ว่าถ้าบิตที่ตรงกันของ X, Y, และ Z จะเป็นอิสระต่อกัน แล้วแต่ละบิตของฟังก์ชั่น G(X, Y, Z), H(X, Y, Z) และ I(X, Y, Z) จะเป็นอิสระต่อกัน เมื่อฟังก์ชั่น H เป็นบิตไวส์ เอ็กซ์ออร์(xor) หรือ พาร์ริตี้ (parity) ฟังก์ชั่นของอินพุต ในขั้นนี้จะใช้ตารางขนาด 64 เอเล้เมนต์ (element) T[1…64] สร้างฟังก์ไซน์ เมื่อ T[i] โดย I เป็นเอเล้เมนต์ของตา

รางซึ่งเท่ากับจำนวนเต็มส่วนของ 4294967296 ครั้ง abs(sin(i)) เมื่อ i มีค่าเป็นในเรเดียน ของเอเลเมนต์ของตารางจะกล่าวในส่วนต่อไป ซึ่งการทำงานจะทำตามนี้

```
/*การกระทำแต่ละ 16 เวิร์ดบล็อก */
For I = 0 to N/16-1 do
/* สำเนาจากบล็อก I ไปยังบล็อก X */
For j = 0 to 15 do
Set X[j] to M[I*16+j]
End /*of loop on j*/
/* เก็บค่า A เท่ากับ AA, Bเท่ากับ BB, C เท่ากับ CC, และ D เท่ากับ DD*/
AA = A
BB = B
CC = C
DD = D
/* รอบที่ 1*/
/* ให้ [abcd k s I ] ใช้แทนโอเปอร์ชั่น a = b + ((a + F(b, c, d) + X[k] + t[I] ) <<< s)*/
/* ทำตาม 16 โอเปอร์เรชั่น*/
[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]
/*รอบที่ 2. */
/* ให้ [abcd k s i] ใช้แทนโอเปอร์ชั่น a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
/* ทำตาม 16 โอเปอร์เรชั่น*/
        [ABCD  1  5 17] [DABC  6  9 18] [CDAB 11 14 19] [BCDA  0 20 20]
        [ABCD  5  5 21] [DABC 10  9 22] [CDAB 15 14 23] [BCDA  4 20 24]
        [ABCD  9  5 25] [DABC 14  9 26] [CDAB  3 14 27] [BCDA  8 20 28]
        [ABCD 13  5 29] [DABC  2  9 30] [CDAB  7 14 31] [BCDA 12 20 32]
        /* รอบที่ 3. */
/* ให้ [abcd k s t] ใช้แทนโอเปอร์เรชั่น a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
/* ทำตาม 16 โอเปอร์เรชั่น*/
        [ABCD  5  4 33] [DABC  8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
        [ABCD  1  4 37] [DABC  4 11 38] [CDAB  7 16 39] [BCDA 10 23 40]
```

[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]

[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]

/* รอบที่ 4. */

/* ให้ [abcd k s t] ใช้แทนโอเปอร์ชั่น a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */

/* ทำตาม 16 โอเปอร์เรชั่น*/

[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]

[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]

[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]

[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]

/* เมื่อการเพิ่มตามนี้ (เป็นการเพิ่มแต่ละตัวของรีจิสเตอร์ทั้ง 4 ตัวโดยค่าที่ก่อนที่บล็อกจะทำการเริ่มทำงาน)*/

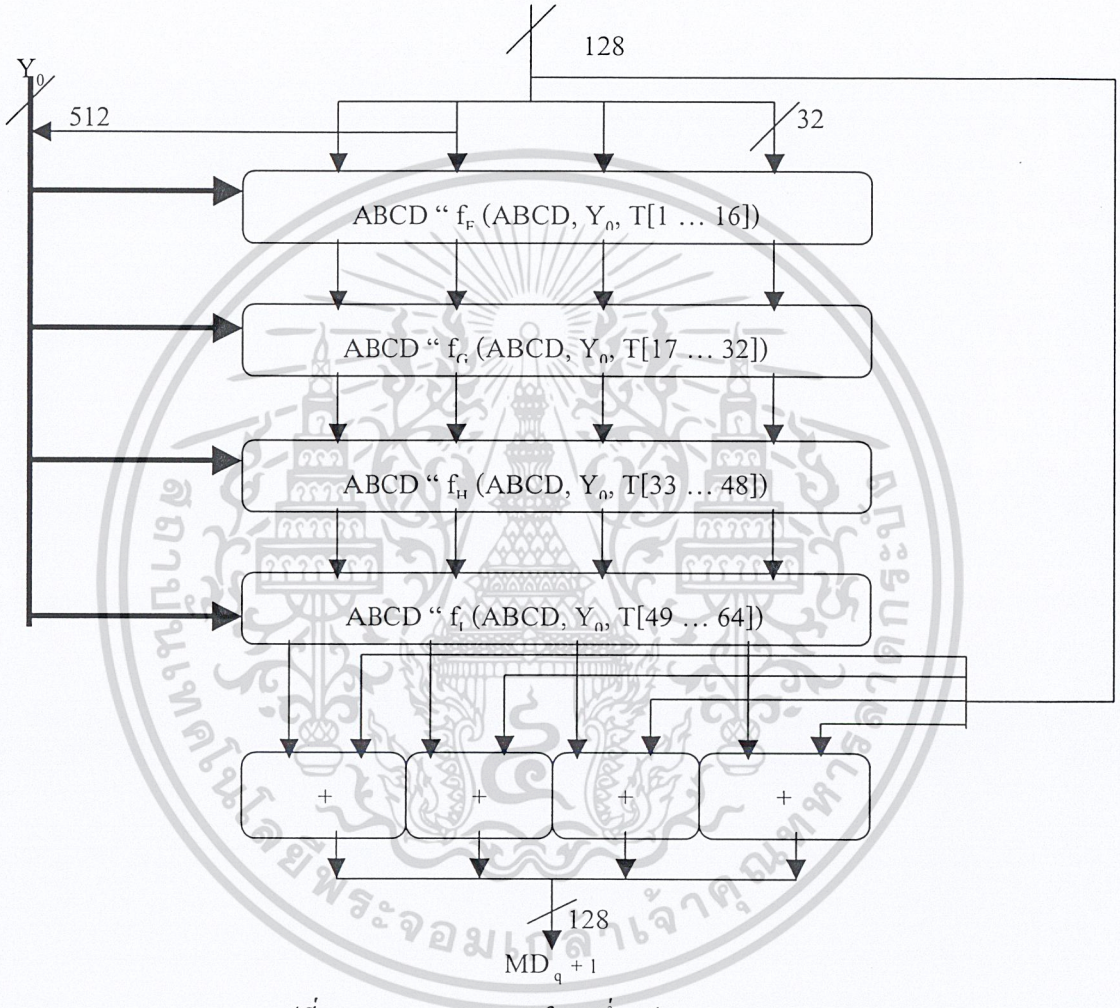
A = A + AA

B = B + BB

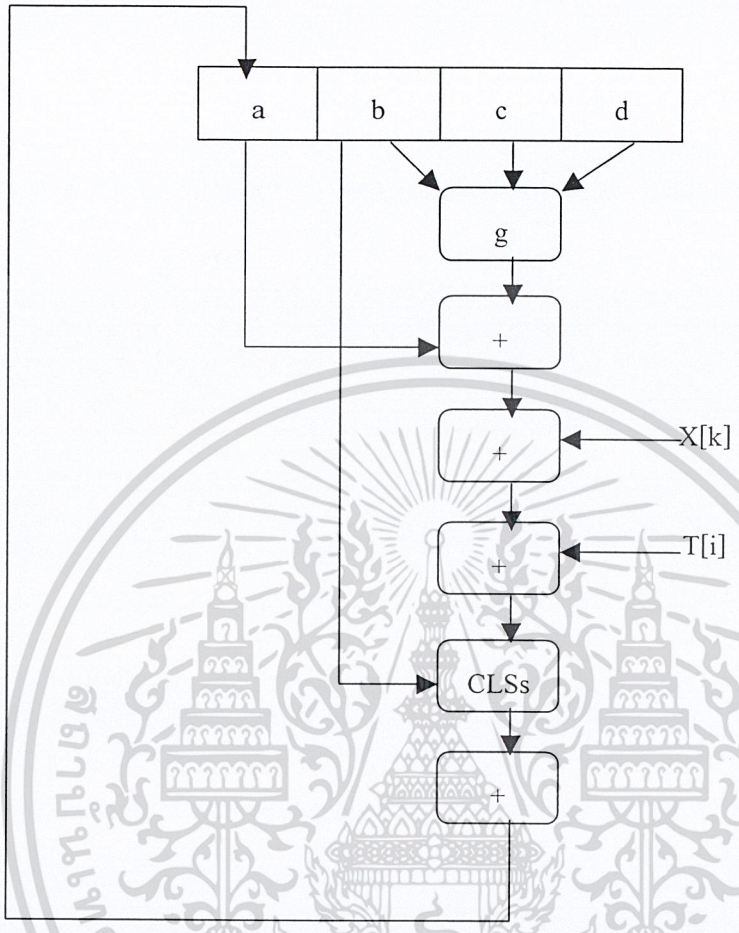
C = C + CC

D = D + DD

end /* รอบของ i */



รูปที่ 4 – 2 กระบวนการทำในหนึ่งบล็อกของ MD5



รูปที่ 4 – 3 แสดงการทำหนึ่งโอเปอร์เรเตอร์ของ MD5 [abcd k s i]

โดยที่ $a \leftarrow b + CLSs(a + g(b, c, d) + X[k] + T[i]$ และ

Round	Primitive function g	G(b, c, d)
F_F	F(b, c, d)	(b . c) v (not(b) . d)
F_G	G(b, c, d)	(b . d) v (c . not(d))
F_H	H(b, c, d)	$b \oplus c \oplus d$
F_I	I(b, c, d)	$c \oplus$ (b . not(d))

ตารางที่ 4 – 1 เงื่อนไขของฟังก์ชั่น

ขั้นตอนที่ 5 เอาท์พุต(output)

แมสเซสดิเจส จะมีเอาท์พุทเป็น A, B, C, D โดยเริ่มที่ไบต์ต่ำของ A และจบด้วยไบต์สูงของ D

4.4 ข้อแตกต่างล่ะหว่างMD4 กับ MD5

ข้อแตกต่างระหว่าง MD4 และ MD5 มีดังนี้

1. มีแค่ 4 รอบ สำหรับการเพิ่ม
2. ในแต่ละขั้นตอนปัจจุบันจะมีการเพิ่มด้วยค่าคงที่เฉพาะ
3. ฟังก์ชั่น g ในรอบที่ 2 จะเปลี่ยนจาก (XY v XZ v YZ) เป็น (XZ v Y not(Z)) และทำเป็นสัดส่วนน้อยที่สุดของ g
4. แต่ละขั้นตอนปัจจุบันจะเพิ่มด้วยค่าจากผลของขั้นตอนที่แล้ว เป็นการนำเสนอที่รวดเร็ว เรียกว่า "avalanche effect"
5. แบบที่อินพุทเวิร์ดที่เข้ามาในรอบที่ 2 รอบที่ 3 จะเปลี่ยน ทำเป็นแพทเทิร์นที่น้อยเหมือนตัวอื่น
6. การเลื่อนรวมในแต่ละรอบจะมีประมาณน้อยสุดและเร็วกว่า "avalanche effect" การเลื่อนในรอบที่แตกต่างจะไม่ซ้ำ

4.5 สรุป

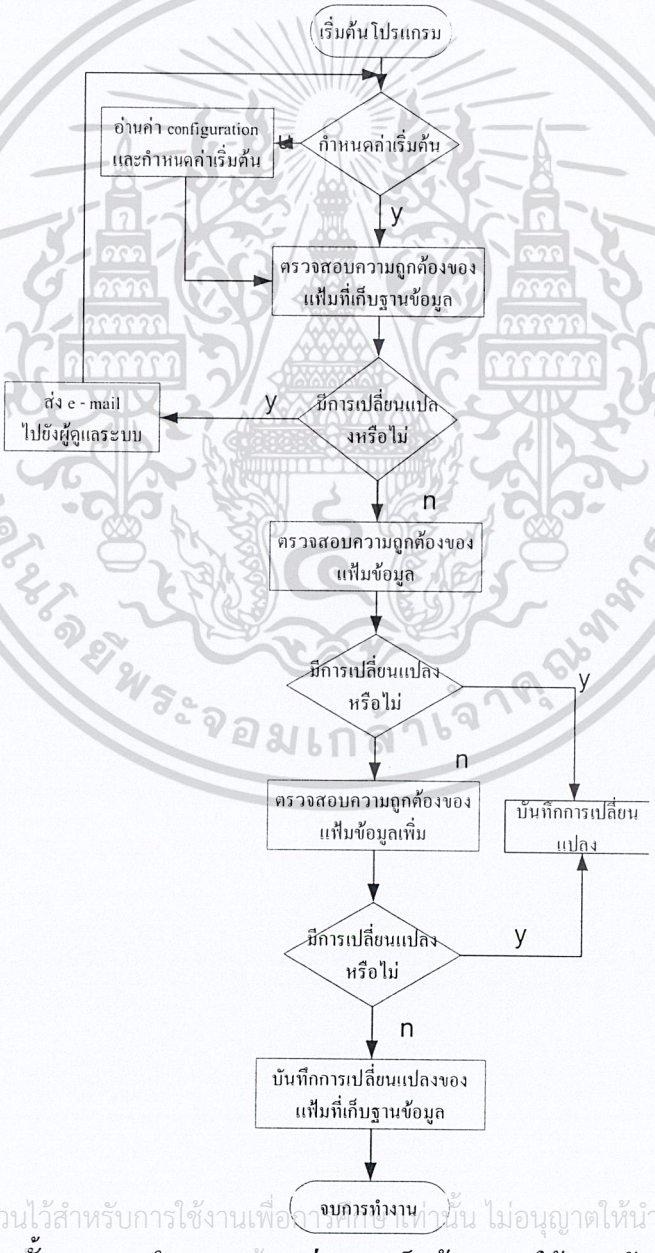
แมสเซสดิเจสลำดับที่5อัลกอริทึมเป็นเครื่องมืออย่างง่าย และเป็นการกำหนดฟริงเกอร์ปริ๊นเบื้องต้น หรือแมสเซสดิเจสของข้อความที่มีความยาวไม่แน่นอน ซึ่งจะทำการเดาค่าที่ค่าที่แตกต่างที่เกิดขึ้นของสองข้อความมีลักษณะที่เหมือนกันของแมสเซสอยู่บนแบบแผนของ 2^64 โอเปอร์เรเตอร์ และ ความแตกต่างที่เกิดขึ้นของข้อความที่แมสเซสดิเจสบนแบบแผนของ 2^128 โอเปอร์เรเตอร์ MD5 ยังระมัดระวังและวิเคราะห์ถึงจุดอ่อน อย่างไรก็ตาม อัลกอริทึมใหม่ที่เกี่ยวข้องและอัลกอริทึมใหม่ที่มีความเกี่ยวข้องตัวใหม่และการวิเคราะห์ความปลอดภัยเป็นการนำเสนอเฉพาะคอร์สเท่ากับเป็นข้อเสนอต่อไปตามลำดับ

บทที่ 5
การวางแผนและการสร้าง

5.1 ภาพรวมและโครงสร้างของโปรแกรม

การออกแบบโปรแกรมตรวจสอบแฟ้มข้อมูลแงออกเป็น 2 ส่วนหลัก ๆ คือ

1. ส่วนของตัวตรวจสอบและวิเคราะห์การเปลี่ยนแปลง
2. ส่วนติดต่อผู้ใช้ระบบ



รูปที่ 5-1 ขั้นตอนการทำงานของการอ่านการเก็บข้อมูลการใช้งานแฟ้มข้อมูล

5.2 การทำงานของแต่ละโปรเซสของการทำงานการอ่านการเก็บข้อมูลการใช้แฟ้มข้อมูล

5.2.1 การอ่าน configuration และการกำหนดค่าเริ่มต้น

ส่วนกำหนดค่าเริ่มต้นของตัวแปรสำหรับแชร์เมมโมรี่และกำหนดค่าเริ่มต้นของฐานข้อมูล

5.2.2 การตรวจสอบความถูกต้องของแฟ้มข้อมูลที่เก็บฐานข้อมูล

เป็นฟังก์ชั่นที่ไว้สำหรับตรวจสอบว่าฐานข้อมูลมีการเปลี่ยนแปลงหรือไม่ หากมีการเปลี่ยนแปลงใด ๆ เกิดขึ้นจะรายงานผลไปยังผู้แลระบบทาง อี- เมล์ โดยทันที และจะหยุดการทำงาน โดยทันที เนื่องจากหากมีการทำงานต่อไปอาจจะเกิดการทำงานที่ผิดพลาดขัดแย้งกับวัตถุประสงค์ของโปรแกรม

5.2.3 การตรวจสอบความถูกต้องของแฟ้มข้อมูลหลัก

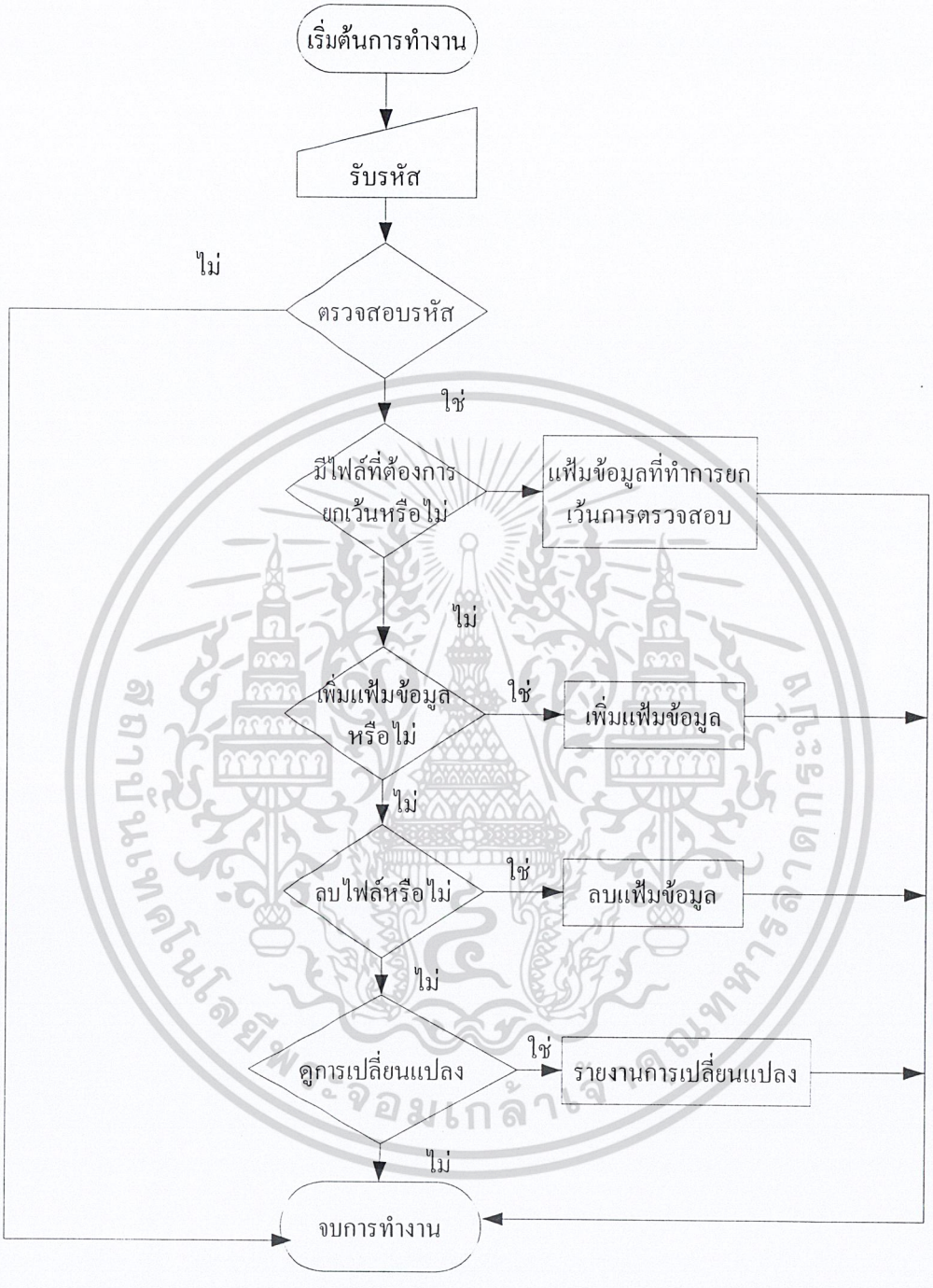
เป็นฟังก์ชั่นที่ไว้สำหรับตรวจสอบการเปลี่ยนแปลงของแฟ้มข้อมูลที่สำคัญ ๆ ซึ่งแฟ้มข้อมูลเหล่านี่จะเป็นแฟ้มข้อมูลหลักที่โปรแกรมจะต้องทำการตรวจสอบไม่สามารถทำการแก้ไขใด ๆ ได้

5.2.4 การตรวจสอบความถูกต้องของแฟ้มข้อมูลที่เพิ่ม

เป็นฟังก์ชั่นที่ไว้สำหรับตรวจสอบการเปลี่ยนแปลงของแฟ้มข้อมูลที่นอกเหนือจากแฟ้มข้อมูลหลักซึ่งการตรวจสอบจะเป็นการดูว่ามีการเปลี่ยนแปลงหรือมาทำการแก้ไข ใด ๆ หรือไม่

5.2.5 การบันทึกการเปลี่ยนแปลงของแฟ้มที่เก็บฐานข้อมูล

เป็นฟังก์ชั่นที่ไว้สำหรับการเก็บข้อมูลของการเปลี่ยนแปลงของแฟ้มของฐานข้อมูลซึ่งเกิดจากการเปลี่ยนแปลงของโปรแกรมโดยเกิดจากการอ่านค่าและเปลี่ยนแปลงค่าในฐานข้อมูล



รูปที่ 5 – 2 ขั้นตอนการทำงานในส่วนการติดต่อกับผู้ดูแลระบบ

5.3 การทำงานในแต่ละส่วนของการทำงานของการติดต่อกับผู้ดูแลระบบ

5.3.1 ตรวจสอบรหัส

เป็นฟังก์ชั่นตรวจสอบว่ารหัสที่รับเข้ามานั้นเป็นรหัสที่ถูกต้องหรือไม่ ถ้าไม่ถูกต้องก็ไม่สามารถที่จะเข้ามาใช้โปรแกรมได้

5.3.2 ตรวจสอบการเพิ่มแฟ้มข้อมูล

เป็นฟังก์ชั่นการตรวจสอบว่ามีการเพิ่มแฟ้มขึ้นมาใหม่หรือไม่ ถ้ามีการเพิ่มแฟ้มข้อมูลขึ้นมาใหม่จะทำการเพิ่มแฟ้มข้อมูลลงสู่แฟ้มข้อมูลที่เป็นแฟ้มฐานข้อมูล

5.3.3 การลบแฟ้มข้อมูล

เป็นฟังก์ชั่นการตรวจสอบว่ามีการลบไฟล์หรือไม่ถ้ามีการลบไฟล์เกิดขึ้นเมื่อมีการลบแฟ้มข้อมูลหรือไดเร็กทอรี่จะทำการลบแฟ้มข้อมูลและไดเร็กทอรี่ออก

5.3.4 ดูการเปลี่ยนแปลง

เป็นฟังก์ชั่นแสดงรายงานการเปลี่ยนแปลงของแฟ้มข้อมูลต่าง ๆ ที่ตัวโปรแกรมทำการตรวจสอบว่ามีการเปลี่ยนแปลงหรือไม่ โดยที่เราสามารถตรวจสอบว่าจะดูการเปลี่ยนแปลงในช่วงเวลาใดก็ได้

5.3.5 ยกเว้นการตรวจสอบแฟ้มข้อมูลที่ต้องการ

เป็นฟังก์ชั่นที่ใช้ในการยกเลิกการตรวจสอบมีแฟ้มข้อมูลที่เราไม่ต้องการที่จะตรวจสอบ เช่น pwd เพราะโดยปกติจะมีการเปลี่ยนแปลงอยู่แล้ว

5.4 การทำงานของโปรแกรม

5.4.1 ส่วนของการใส่อินพุต

คำสั่งที่ใช้ในที่จะเพิ่มหรือลบแฟ้มข้อมูลหรือไดเร็กทอรี่มีรูปแบบดังนี้

promt # isagcheck <optional parameter> -<mode> -<filename>

<optional parameter> ประกอบไปด้วย 2 พารามิเตอร์ คือ

add สำหรับเพิ่มแฟ้มข้อมูลหรือไดเร็กทอรี่ที่ต้องการตรวจสอบเพิ่มเติม

del สำหรับลบแฟ้มข้อมูลหรือไดเร็กทอรี่ที่ไม่ต้องการตรวจสอบ

-<mode> ประกอบด้วย 2 โหมดการทำงาน คือ

-f เป็นตัวบอกว่าเป็นแฟ้มข้อมูล

-d เป็นตัวบอกว่าเป็นไดเร็กทอรี่

-<filename>

เป็นชื่อแฟ้มข้อมูลหรือไดร์กทอรี่

ตัวอย่าง

#isagcheck add –f /home/rave/.bash_history

เป็นตรวจสอบแฟ้มข้อมูลชื่อ /home/rave/.bash_history เพิ่มจากเดิม

หรือ

#isagcheck del -f /home/rave/.bash_history

เป็นลบการตรวจสอบแฟ้มข้อมูลชื่อ /home/rave/.bash_history

5.4.2 ส่วนที่ยกเว้นการตรวจสอบ

คำสั่งที่ใช้ในการยกเว้นทั้งหมดมีรูปแบบดังนี้

```
promt # isagcheck <optional parameter> -<mode> <filename>
```

<optional parameter> ประกอบไปด้วย 2 พารามิเตอร์ คือ

drop ยกเว้นการตรวจสอบแฟ้มข้อมูลหรือไดเร็กทอรี่

active ยกเลิกการทำงานของพารามิเตอร์ drop

-<mode> ประกอบไปด้วย 2 โหมดการทำงาน คือ

-f เป็นการบอกว่าเป็นแฟ้มข้อมูล

-d เป็นการบอกว่าเป็นไดเร็กทอรี่

<filename>

เป็นชื่อแฟ้มข้อมูลหรือไดเร็กทอรี่ที่ต้องการตรวจสอบ

ตัวอย่าง

#isagcheck drop -f /etc/passwd

เป็นการยกเว้นการตรวจสอบ แฟ้มข้อมูลชื่อ /etc/passwd

#isagcheck active –f /etc/passwd

เป็นการยกเลิกการทำงานของ drop ของ /etc/passwd

5.4.3 ส่วนของเอาต์พุต

คำสั่งในการดูการเปลี่ยนแปลงของแฟ้มข้อมูลแบ่งออกเป็น 2 ส่วนหลัก ๆ คือ การดูการเปลี่ยนแปลงโดยเลือกดูการเปลี่ยนแปลง และ การดูการเปลี่ยนแปลงของทุก ๆ การเปลี่ยนแปลง

5.4.3.1 การดูการเปลี่ยนแปลงโดยการเลือกดูการเปลี่ยนแปลง

```
promt # isagcheck  <optional  parameter>  -<mode>  <log>  <date>  <time>
```

<optional parameter> ประกอบด้วย 1 พารามิเตอร์ คือ

report

-<mode> เป็นการเลือกดูส่วนที่เปลี่ยนแปลง ประกอบไปด้วย 4 โหมดการทำงาน คือ

-h เลือกดูช่วงเวลาที่ต้องการจะดู

-d เลือกวันที่ต้องการดู

-m เลือกเดือนที่ต้องการ

-y เลือกปีที่ต้องการดู

<log> เป็นตัวบ่งบอกว่าจะเลือกดูการเปลี่ยนแปลงแบบไหน ประกอบไปด้วย 4 ตัว คือ

acc เป็นการดูการเข้าใช้แฟ้มข้อมูลนั้น ๆ โดยไม่มีการแก้ไขแต่อย่างไร

sta เป็นการดูการแก้ไข status ของแฟ้มข้อมูลนั้น ๆ โดยไม่มีการแก้ไขเนื้อข้อมูลนั้น ๆ

mod เป็นการดูการเข้าไปแก้ไขข้อมูลของแฟ้มข้อมูลนั้น ๆ

ino เป็นการดูการเปลี่ยนแปลงของไอโหนดของแฟ้มข้อมูลโดยที่เนื้อแฟ้มข้อมูลอาจไม่มีการเปลี่ยนแปลงก็ได้

<date> เป็นตัวบ่งบอกว่าจะเลือกดูวันเดือนปีไหน โดย

ถ้าหาก <mode> เป็น –h จะมีรูปแบบคือ 10/01/2003

ถ้าหาก <mode> เป็น –d จะมีรูปแบบคือ 10/01/2003

ถ้าหาก <mode> เป็น –m จะมีรูปแบบคือ 01/2003

ถ้าหาก <mode> เป็น –y จะมีรูปแบบคือ 2003

<time> เป็นตัวบ่งชี้ว่าจะต้องการดูตั้งแต่ช่วงเวลาไหนในวันปัจจุบัน

เช่น ต้องการดูตั้งแต่เวลา 6 นาฬิกา ถึง 12 นาฬิกา คือ 6 – 12

ตัวอย่าง

#isagcheck report - d acc 10/01/2003

#isagcheck report - h mod 10/01/2003

5.4.3.2 การดูการเปลี่ยนแปลงของทุก ๆ การเปลี่ยนแปลง
จะมีรูปแบบ ดังนี้

```
promt # isagcheck  <optional parameter>  -a <date>  <time>
```

<optional parameter> ประกอบด้วย report

-a
เป็นตัวบ่งบอกว่าเป็นการดูการเปลี่ยนแปลงทั้งหมด

<date> เป็นตัวบ่งบอกว่าจะเลือกดูวันเดือนปีไหน โดย
ถ้าหาก <mode> เป็น –h จะมีรูปแบบคือ 10/01/2003
ถ้าหาก <mode> เป็น –d จะมีรูปแบบคือ 10/01/2003
ถ้าหาก <mode> เป็น –m จะมีรูปแบบคือ 01/2003
ถ้าหาก <mode> เป็น –y จะมีรูปแบบคือ 2003

<time> เป็นตัวบ่งชี้ว่าจะต้องการดูตั้งแต่ช่วงเวลาไหนในวันปัจจุบัน
เช่น ต้องการดูตั้งแต่เวลา 6 นาฬิกา ถึง 12 นาฬิกา คือ 6 – 12

ตัวอย่าง

#isagcheck report - a - m 01/2003

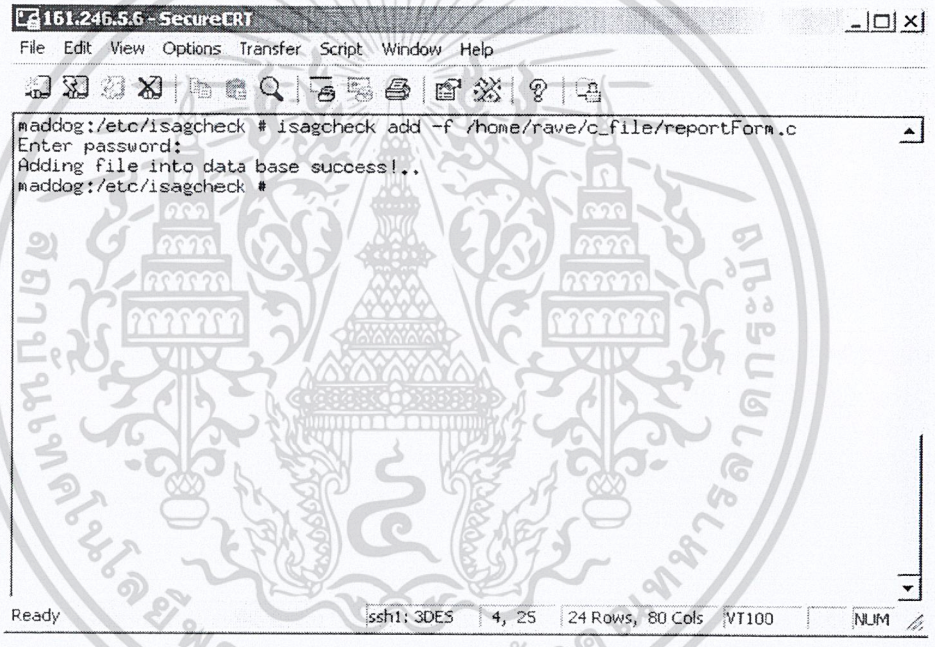
#isagcheck report - h - h 10/01/2003 6-12

บทที่ 6

การทดลองและผลการทดลอง

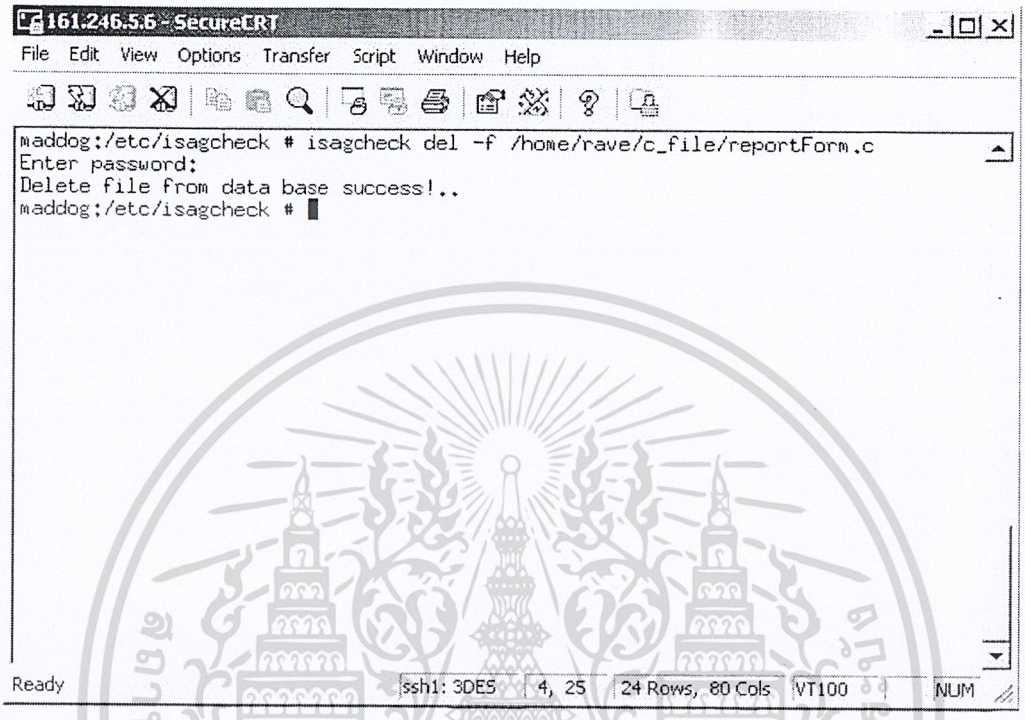
6.1 ทดลองการทำงานของส่วนของอินพุต

การทดลองเพิ่มแฟ้มข้อมูลที่จะทำการตรวจสอบ



รูปที่ 6 – 1 การทดลองการเพิ่มแฟ้มข้อมูลที่จะตรวจสอบ

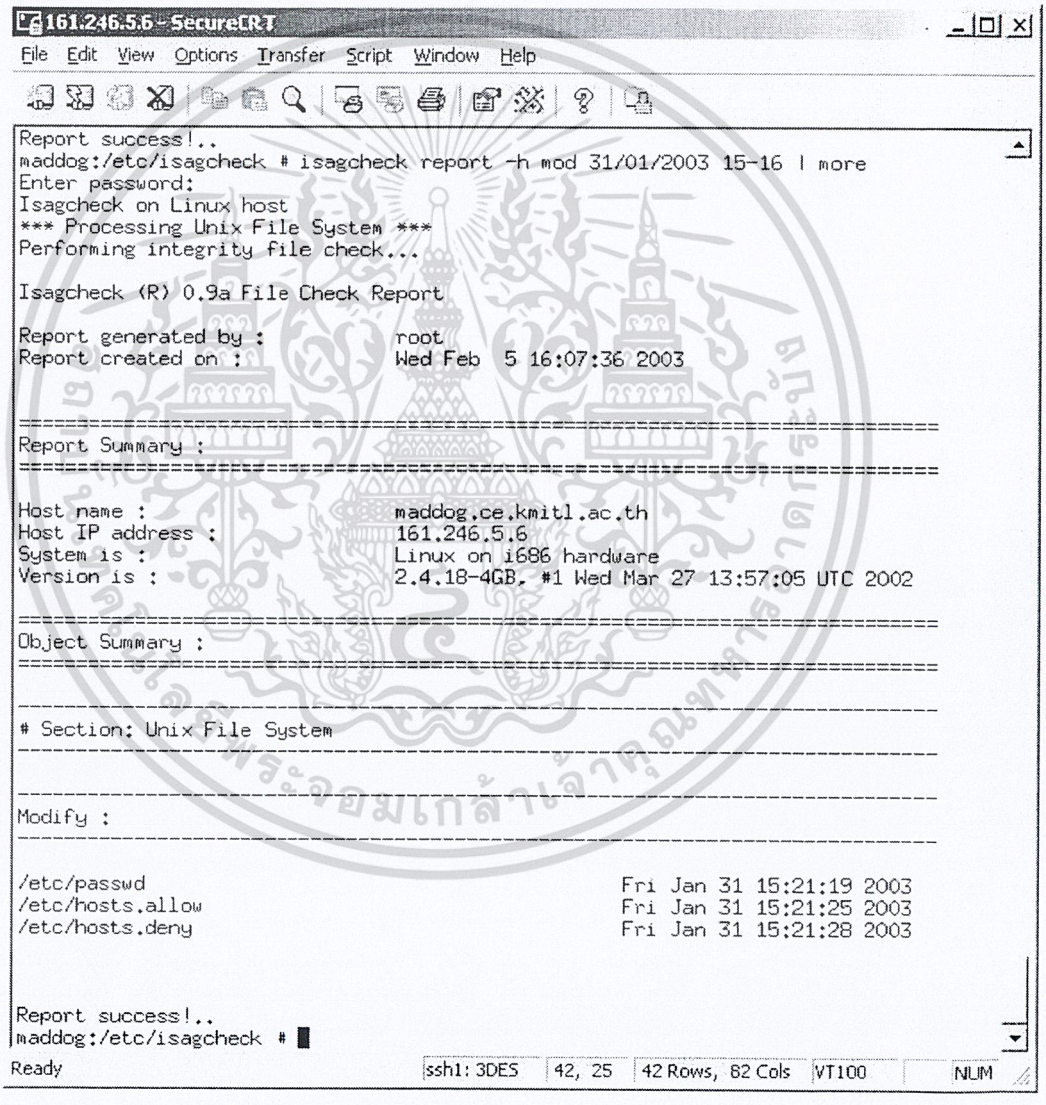
การทดลองลบแฟ้มข้อมูลออกจากการตรวจสอบ



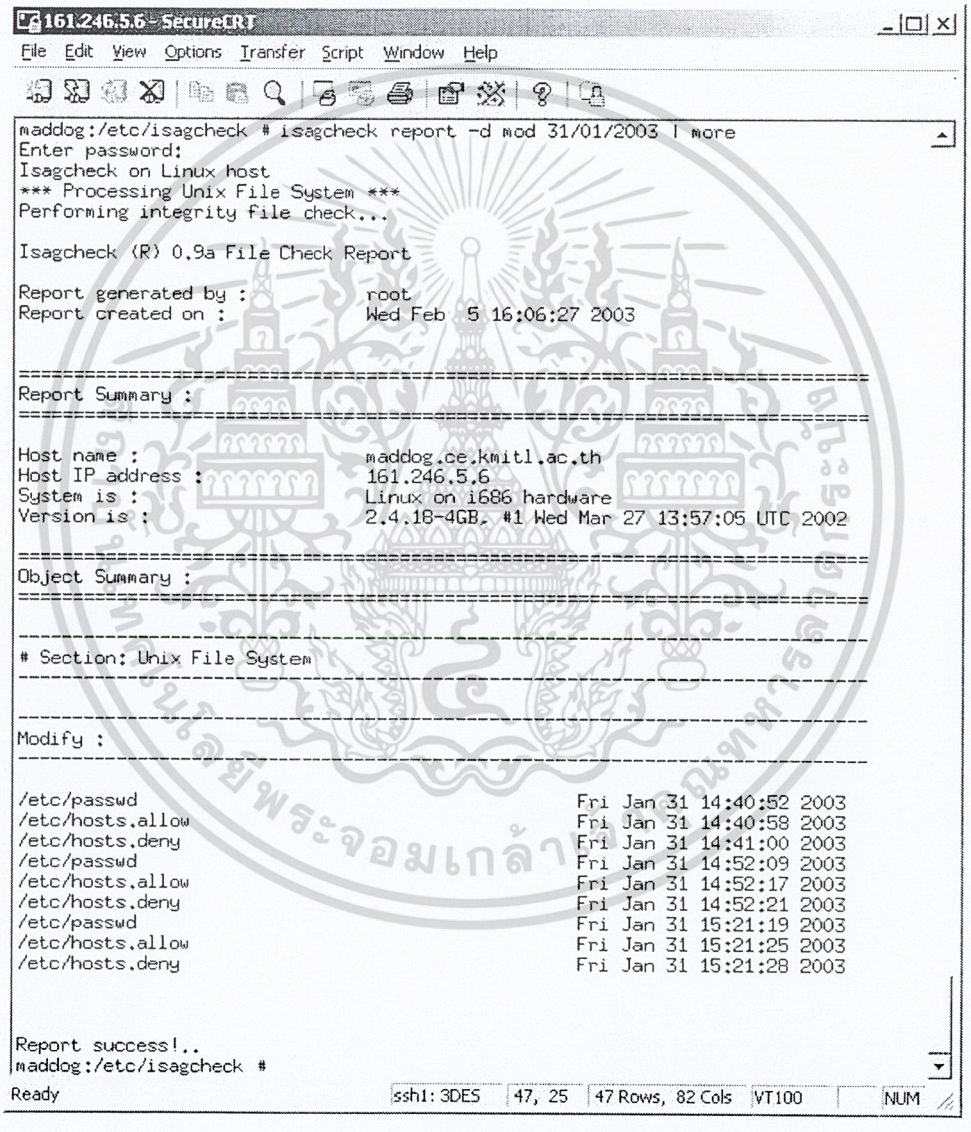
รูปที่ 6 – 2 การลบแฟ้มข้อมูลออกจากการตรวจสอบ

6.2 ทดลองการรายงานการเปลี่ยนแปลง

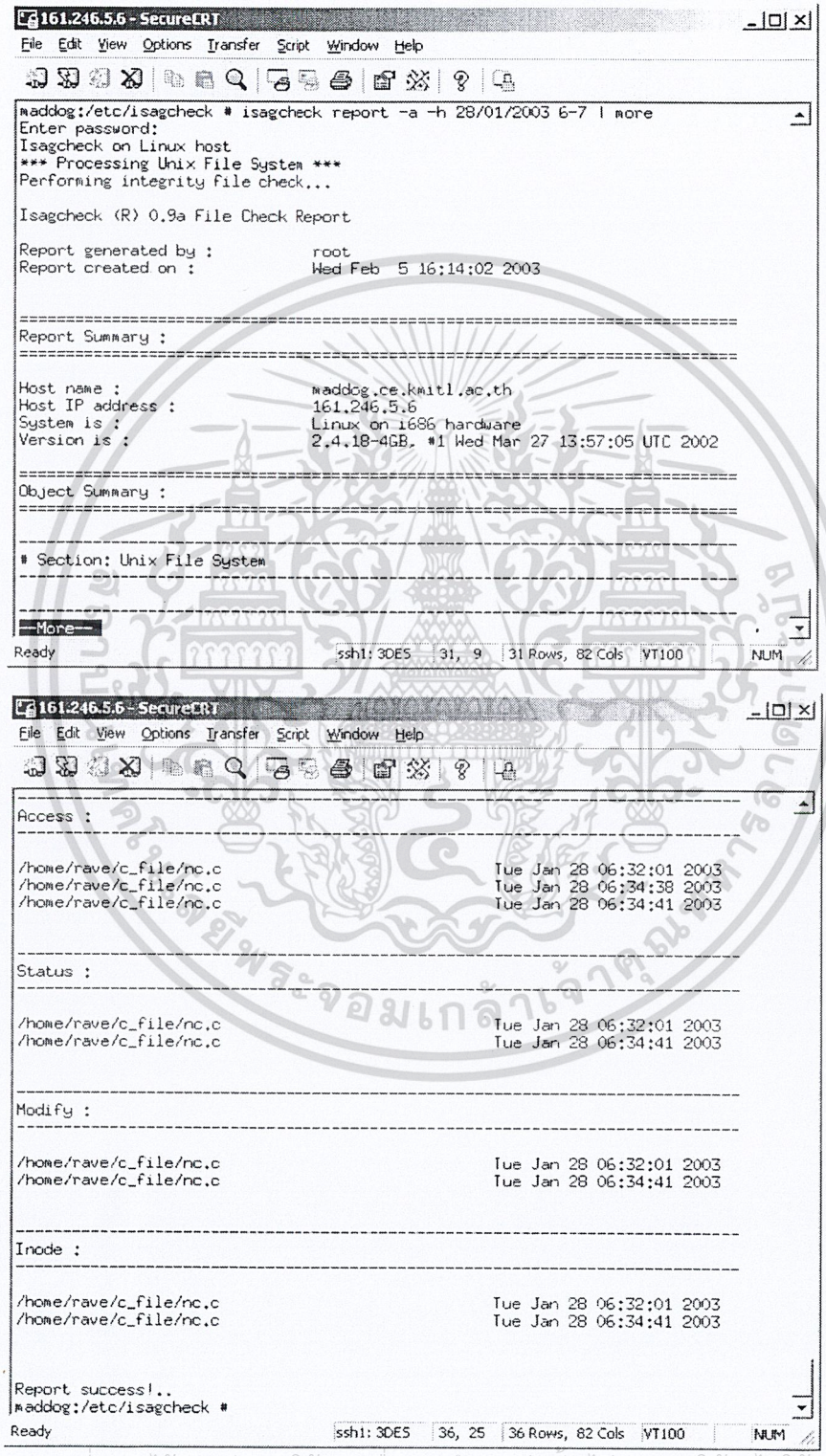
เป็นการทดสอบการเปลี่ยนแปลงของแฟ้มข้อมูลว่าต้องการจะดูรายงานการเปลี่ยนแปลงของแฟ้มข้อมูลในลักษณะใดบ้าง โดยที่สามารถที่จะทำการเลือกดูการเปลี่ยนแปลงได้ว่าต้องการดูส่วนใดบ้าง



รูปที่ 6 – 3 ดูรายงานการเปลี่ยนแปลงในวันที่ 31/01/2003 เวลา 15 –16 นาฬิกา



รูปที่ 6 – 4 ดูรายการเปลี่ยนแปลงในวันที่ 31/01/2003



รูปที่ 6 – 5 แสดงการเปลี่ยนแปลงทั้งหมดที่เกิดขึ้น

บทที่ 7
วิจารณ์และสรุป

7.1 ผลการทำงานของโปรแกรม

โครงงานนี้สามารถที่จะตรวจสอบส่วนของแฟ้มข้อมูลในระบบของยูนิกซ์ในเชิงรับคือสามารถที่จะตรวจสอบการเปลี่ยนแปลงของแฟ้มข้อมูลหรือตรวจสอบการทำงานของแฟ้มข้อมูลว่ามีการเปลี่ยนแปลงหรือไม่ โดยสามารถที่จะทำการเลือกแฟ้มข้อมูลที่จะทำการตรวจสอบ หรือว่าจะมีการยกเว้นแฟ้มข้อมูลใดบ้าง แล้วสามารถที่จะดูรายงานการเปลี่ยนแปลงได้ว่ามีการเปลี่ยนแปลงอย่างไรบ้าง โดยสามารถที่จะเลือกดูได้ว่าจะดูการเปลี่ยนแปลงทั้งหมดหรือดูตามช่วงเวลา วัน เดือน และปี ที่ต้องการได้

7.2 แนวทางการพัฒนาต่อ

การพัฒนาโครงงานนี้มีความสามารถที่จะทำการตรวจสอบว่าจะมีการแก้ไขแฟ้มข้อมูลหรือมีการกระทำใด ๆ กับแฟ้มข้อมูลนั้นนั้นได้เกือบที่จะครอบคลุม เหลือเพียงแต่ยังไม่สามารถตรวจสอบได้ว่ามีแฟ้มข้อมูลใดเพิ่มขึ้นมา หรือแฟ้มข้อมูลใดถูกลบไปจากระบบบ้าง หากมีการพัฒนาต่อควรที่จะเพิ่มความสามารถด้านนี้เข้าไปด้วย ในด้านการตรวจสอบฐานข้อมูลที่เก็บไว้เปิดเปรียบเทียบยังไม่ได้เข้ารหัสข้อมูลไว้แต่สามารถที่จะตรวจสอบว่ามีการกระทำใด ๆ กับฐานข้อมูลนั้นได้หรือไม่

แนวทางการพัฒนาต่อควรที่จะทำการเข้ารหัสฐานข้อมูลได้อีกด้วย

7.3 ปัญหาที่พบในการทำโครงงาน

ปัญหาใหญ่ ๆ ที่พบและหาวิธีแก้ไม่ได้คือ เมื่อเราจะเข้ารหัสแฟ้มข้อมูลใด ๆ ถ้าเรามีคีย์ที่ใช้ในการเข้ารหัส และคีย์นั้นจะต้องฝังอยู่ใน source code แล้วหากเรา open source code จะทำให้ผู้ที่เข้ามาดู source code จะสามารถทราบและเข้าฟังก์ชั่นที่เราได้ใช้ไปแก้ไขได้ทันที ซึ่งในการทำโครงงานนี้ได้ทำการตรวจสอบว่ามีผู้ใดนอกจากโปรแกรมของเราเองมาใช้ฐานข้อมูลหรือ เข้ามาแก้ไขก็จะทำการรายงานผลผ่าน อีเมล์ไปยังผู้ดูแลระบบ

บรรณานุกรม

หนังสืออ้างอิง

1. David A. Curry, “Using C on the UNIX System”, O’Reilly & Association, Inc.
2. Chris Brown, “UNIX Distributed Programing” ,PRENTICE-HALL
3. Maurice J. Bach, “THE DESIGN OF THE UNIX OPERATING SYSTEM”, PRINTICE-HALL
4. Moshe Bar, “ Linux Internals”, McGraw-Hill
5. Niel Matthew & Rechard Stones, “BEGING Linux Programming”, WROX PRESS Ltd.
6. สันติ ศรีลาศักดิ์ & วรวุฒิ เที่ยงธรรม, “เจาะประเด็นงานเขียนโปรแกรมบนลีนุกซ์ ” , OFFSET PRESS COMPANY LIMITED

เว็บไซต์อ้างอิง

1. www.linux.org
2. www.tripwire.org
3. www.hunter.com/docs/rfc/rfc1321.html
4. www.ececs.uc.edu/~zxu/reports/filesystem.htm