การพัฒนาระบบปฏิบัติการมินิกซ์

MINIX OPERATING SYSTEM ENHANCEMENT

โดย

นาย ชูกิจ	วนาธรรม	301062
นาย วีรศักดิ์	วิทวัสกุล	301264
นาย สราวุธ	อยู่วิทยา	301304
นาย อาทิตย์	จิตต์จุฬานนท์	301360

อาจารย์ที่ปรึกษา

ดร. บุญธีร์ เครือตราชู

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า วิทยาเขตเจ้าคุณทหาร ลาดกระบัง

ปีการศึกษา 2533

ปีการศึกษา 2533

การพัฒนาระบบปฏิบัติการมินิกซ์

จัดทำโดย

นาย ชูกิจ	วนาธรรม	301062
นาย วีรศักดิ์	วิทวัสกุล	301264
นาย สราวุฒิ	อยู่วิทยา	301304
นาย อาทิตย์	จิตต์จุฬานนท์	301360

อาจารย์ที่ปรึกษา

ดร. บุญธีร์ เครือตราชู

ปริญญานิพนธ์ปีการศึกษา 2533

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

เรื่อง การพัฒนาระบบปฏิบัติการมินิกซ์

คณะผู้จัดทำ

1. นาย ชุกิจ วนาธรรม 301062
2. นาย วีรศักดิ์ วิทวัสกุล 301264
3. นาย สราวุฒิ อยู่วิทยา 301304
4. นาย อาทิตย์ จิตต์จุฬานนท์ 301360

..... อาจารย์ที่ปรึกษา

(ดร.บุญธีร์ เครือตราชู)

การพัฒนาระบบปฏิบัติการมินิกซ์

นาย ชูกิจ วนาธรรม
นาย วีรศักดิ์ วิทวัสกุล
นาย สราวุฒิ อยู่วิทยา
นาย อาทิตย์ จิตต์จุฬานนท์

ดร. บุญธีร์ เครือตราชู

ปีการศึกษา 2533

บทคัดย่อ

วิทยานิพนธ์นี้ การพัฒนาระบบปฏิบัติการมินิกซ์ มีจุดประสงค์เพื่อที่จะศึกษาการพัฒนาระบบปฏิบัติการที่มีอยู่แล้ว ให้มีความสามารถสูงขึ้น โดยโครงงานนี้เป็นการพัฒนาระบบปฏิบัติการมินิกซ์ ซึ่งเป็นระบบปฏิบัติการแบบมัลติทาสก์ซึ่งทำงานบนเครื่องไอบีเอ็มพีซี เพื่อให้สามารถทำงานกับภาษาไทยได้ โดยจะมีคำสั่งเรียกระบบให้โปรแกรมได้เรียกใช้ด้วย

การดำเนินงานของโครงงานนี้ แบ่งการดำเนินงานเป็นสองส่วน โดยในส่วนแรกเป็นการศึกษาการทำงาน และซอร์สโค้ดภาษาซี และแอสเซมบลีของมินิกซ์ หลังจากศึกษาจนเข้าใจแล้ว จึงเริ่มทำงานส่วนที่สอง คือ วางเค้าโครงงาน การพัฒนา และทำการเปลี่ยนแปลง และเพิ่มเติมโปรแกรม พัฒนาให้ทำงานในกราฟิกโหมด แสดงผลภาษาไทย รับคีย์บอร์ดภาษาไทย สร้างคำสั่งเรียกระบบ และสร้างโปรแกรมตัวอย่าง ตามลำดับ

MINIX OPERATING SYSTEM ENHANCEMENT

Chookit Vanatham

Weerasak Witthawaskul

Saravoot Yoovidhya

Arthid Chitchulanon

Dr. Boontee Cruatrachu Advisor

1990

Abstract

The object of this thesis,MINIX OPERATING SYSTEM ENHANCEMENT , is to learn how to develope existing operating system in order to obtain higher performance. In this project, we 've chosen to enhance the IBM-PC multitasking operating system, MINIX, by adding thai system into its kernel. Moreover, some thai system calls 're also provided.

The process of this project 's divided into two parts. In the first part, we study how MINIX works and how it 's implement by means of more than twelve thousand lines of C and assembly source codes. Having clearly understood, the second part begins by designing, modifying-adding codes, changing MINIX into graphic mode, making it display and receive thai language , building some system calls and writing example program, respectively.

สารบาญ

หน้า

ภาคที่หนึ่ง

หน้า

ภาคที่สอง

หน้า

หน้า

ภาคที่ ๑

บทที่ 1

บทนำและประวัติของระบบปฏิบัติการมินิกซ์

บทนำ

วิชาระบบปฏิบัติการ เป็นวิชาหลักที่นักศึกษาคณะวิศวกรรมศาสตร์ สาขาคอมพิวเตอร์จำเป็นต้องศึกษา เนื่องจากระบบปฏิบัติการนั้น เป็นโปรแกรมที่ซับซ้อน มีหลักการ ทฤษฎีมากมาย ซึ่งผู้ศึกษาสามารถนำไปประยุกต์ใช้ได้ในหลาย ๆ ด้าน และเพราะความซับซ้อนของโปรแกรมอีกเช่นกันที่ทำให้การศึกษาระบบปฏิบัติการโดยส่วนใหญ่ เป็นแต่ในภาคทฤษฎีเท่านั้น ทำให้ผู้ศึกษาบางครั้งอาจมองไม่เห็นภาพจริง ๆ ของระบบ ดังนั้น ระบบปฏิบัติการมินิกซ์ จึงถูกพัฒนาขึ้นเพื่อการศึกษาระบบปฏิบัติการภาคปฏิบัติ

มินิกซ์เป็นระบบปฏิบัติการแบบมัลติทาสก์ กล่าวคือสามารถทำงานหลาย ๆ งานพร้อมกัน ซึ่งมีซอร์สโค้ดประกอบมาพร้อมกับหนังสือ " OPERATING SYSTEM Design and Implementation " โดยซอร์สโค้ดทั้งหมดจะมีให้ในแผ่นดิสก์ ดังนั้นผู้ศึกษาระบบปฏิบัติการจึงสามารถศึกษาจากซอร์สโค้ดจริง ๆ อีกทั้งยังสามารถทดลองดัดแปลงการทำงานของระบบได้อีกด้วย ซึ่งเป็นการศึกษาภาคปฏิบัติที่ดี โดยไม่จำเป็นต้องเขียนระบบปฏิบัติการใหม่เองทั้งหมด ซึ่งจะเป็นงานที่ยาก และกินเวลามาก

โครงงานการพัฒนาระบบปฏิบัติการมินิกซ์นี้ มีจุดประสงค์ในการพัฒนาเพิ่มความสามารถให้แก่ระบบปฏิบัติการมินิกซ์ โดยการจะทำให้มินิกซ์สามารถทำงานภาษาไทยได้ รวมทั้งมีคำสั่งเรียกระบบให้โปรแกรมอื่น ๆ ที่ทำงานภายใต้มินิกซ์เรียกใช้ และเนื่องจากความต้องการให้สามารถแสดงผลภาษาไทยได้ จึงต้องเปลี่ยนโหมดการแสดงผลให้อยู่ในโหมดกราฟิคทั้งหมด อีกทั้งยังมีโปรแกรมตัวอย่างที่พัฒนาขึ้นภายใต้มินิกซ์ใหม่ที่พัฒนาเสร็จแล้ว เพื่อที่จะแสดงการเขียนโปรแกรมที่มีการใช้คำสั่งเรียกระบบเกี่ยวกับภาษาไทย ได้แก่ อิดิเตอร์ที่สนับสนุนภาษาไทย และโปรแกรมที่จะช่วยในการใช้งานมินิกซ์ให้ง่ายขึ้น ได้แก่ โปรแกรมเชลล์

อย่างไรก็ตาม จุดประสงค์ของโครงงานนี้ มิได้เน้นที่จะพัฒนาระบบภาษาไทยให้ใช้งานได้เท่าเทียมกับระบบภาษาไทยที่มีอยู่ในท้องตลาดปัจจุบัน แต่เน้นการพัฒนาระบบภาษาไทยให้เป็นส่วนหนึ่งของระบบปฏิบัติการเท่านั้น

โครงสร้างของปริญญานิพนธ์ฉบับนี้ จะแบ่งออกเป็นสองภาคด้วยกัน คือ ภาคแรกเป็นการศึกษาถึงรายละเอียดต่างๆของระบบปฏิบัติการมินิกซ์เดิมว่า มีการจัดโครงสร้างของระบบเป็นอย่างไร โดยมีเนื้อหาการศึกษาออกเป็นสามส่วนใหญ่ด้วยกัน คือ ส่วนเคอร์นัล ซึ่งจะพูดถึงโครงสร้างของโปรเซส การติดต่อกันระหว่างโปรเซส การจัดสรรเวลาให้แก่โปรเซส การจัดการกับอินเทอร์รัพท์ การติดต่อกับอุปกรณ์อินพุท/เอาท์พุท สำหรับส่วนที่สอง คือ ส่วนจัดการกับหน่วยความจำ จะกล่าวถึง การจัดการกับหน่วยความจำ ข่าวสาร และสัญญาณต่างๆ เช่น การฟอร์คโปรเซส เป็นต้น และส่วนสุดท้าย คือ ส่วนระบบไฟล์ ซึ่งจะพูดถึงโครงสร้างของระบบไฟล์ การจัดการข่าวสาร การทำบล็อคแคช และซิสเต็มคอลล์ต่างๆ

การศึกษารายละเอียดต่างๆในภาคแรก จะเน้นไปที่ส่วนเคอร์นัลซึ่งเป็นส่วนที่รวมการติดต่อกับอุปกรณ์อินพุท/เอาท์พุท เช่น จอภาพ แป้นพิมพ์ เป็นต้น ซึ่งรายละเอียดที่ศึกษาในภาคแรก จะเป็นข้อมูลในการพัฒนาระบบเป็นระบบปฏิบัติการมินิกซ์ที่สามารถรับข้อมูลภาษาไทยได้ โดยรายละเอียดการแก้ไขรวมไปถึงจุดต่างๆที่แก้ไข จะกล่าวอธิบายไว้ในภาคที่สอง ซึ่งเป็นภาคปฏิบัติ โดยจะกล่าวถึงวิธีการแก้ไขซอร์สโปรแกรม การสร้างแผ่นบู๊ตดิสค์ที่มีฟอนต์ภาษาไทยรวมอยู่ด้วย และตัวอย่างโปรแกรมประยุกต์ใช้งานต่างๆ

มินิกซ์ คืออะไร

มินิกซ์ (MINIX) เป็นระบบปฏิบัติการหรือ Operating System (โอเอส) ที่มีความสามารถในการทำงานหลายๆงาน (multitasking) และหลายผู้ใช้ (multiuser) โดยมีการใช้งานคล้ายคลึงกับการใช้ระบบดำเนินงานยูนิกซ์ (UNIX) เมื่อมองจากมุมมองของผู้ใช้งาน แต่มีโครงสร้างภายในที่แตกต่างกัน โดยชื่อมินิกซ์นั้นย่อมาจาก Mini-Unix ซึ่งหมายถึงเป็นระบบดำเนินงานยูนิกซ์ขนาดเล็กนั่นเอง

สำหรับตัวซอฟท์แวร์มินิกซ์ จะประกอบด้วยโอเอสมินิกซ์, ซอร์สโค้ด (source code) ที่เขียนขึ้นจากภาษาซีและแอสเซมบลี้, คอมไพเลอร์ภาษาซี, แอสเซมเบลอร์ รวมทั้งโปรแกรมคอมมานด์ต่างๆ ซึ่งทั้งหมดจัดเก็บอยู่ในโอเอสมินิกซ์ การพัฒนาโอเอสหรือโปรแกรมประยุกต์ใช้งาน (application) ก็สามารถทำการพัฒนาบนมินิกซ์ได้โดยตรง

ประวัติของมินิกซ์

เนื่องมาจากในยุคที่มีการพัฒนาโอเอสยูนิกซ์ใหม่ๆ ซอร์สโค้ดยังไม่เป็นที่ปกปิด จึงสามารถนำตัวโปรแกรมมาใช้ศึกษาการทำงานของโอเอสตามสถานศึกษาได้ ภายใต้การขออนุญาตจากบริษัทเอทีแอนด์ที (AT&T) เพื่อให้นักศึกษาสามารถมองเห็นภาพพจน์ของการทำงานของโอเอสได้ดีขึ้นกว่าที่จะศึกษาแต่เพียงทฤษฎีอย่างเดียว แต่ในภายหลังโอเอสยูนิกซ์เริ่มกลายเป็นผลิตภัณฑ์ทางการค้าที่มีความนิยมสูงมาก จึงได้เริ่มมีการห้ามนำซอร์สโค้ดของโปรแกรมมาศึกษากันในคอร์สเรียนอีกต่อไป รวมทั้งไม่เปิดเผยซอร์สโค้ดของโอเอสยูนิกซ์เวอร์ชันใหม่ๆที่พัฒนาต่อจากเดิม เพื่อป้องกันความลับทางการค้ามิให้รั่วไหล ทำให้การศึกษาโอเอสในชั้นเรียนจะทำได้แต่เพียงพูดถึงทฤษฎีเท่านั้น ซึ่งตัวทฤษฎีเองก็มิได้เน้นถึงการสร้างระบบจริง และบางหัวข้อที่มีทฤษฎีอ้างถึงน้อย ก็แทบจะไม่ได้กล่าวถึงรายละเอียดมากนัก เช่น การติดต่อกับอุปกรณ์ไอโอ, ระบบไฟล์ (file system) เป็นต้น จึงทำให้เกิดที่มาของระบบปฏิบัติการมินิกซ์ ซึ่งออกแบบและสร้างโดยนาย Andrew S. Tanenbaum แห่งมหาวิทยาลัย Vrije ประเทศเนเธอร์แลนด์

ผู้สร้างมินิกซ์จึงได้ออกแบบโอเอสขึ้นใหม่ ให้มีความสามารถในระดับที่เทียบเท่า (compatibility) กับโอเอสยูนิกซ์ซึ่งเป็นโอเอสที่มีความนิยมใช้แพร่หลาย โดยทำการเขียนโปรแกรมมินิกซ์ขึ้นมาใหม่ทั้งหมด เพื่อหลีกเลี่ยงปัญหาลิขสิทธิ์ สำหรับนำมาใช้ศึกษาเป็นตัวอย่างของโอเอสที่สร้างขึ้นมาเพื่อให้เห็นจริง ทั้งการใช้งานและโครงสร้างภายในของระบบ เพื่อนำมาศึกษาและพัฒนาต่อไป

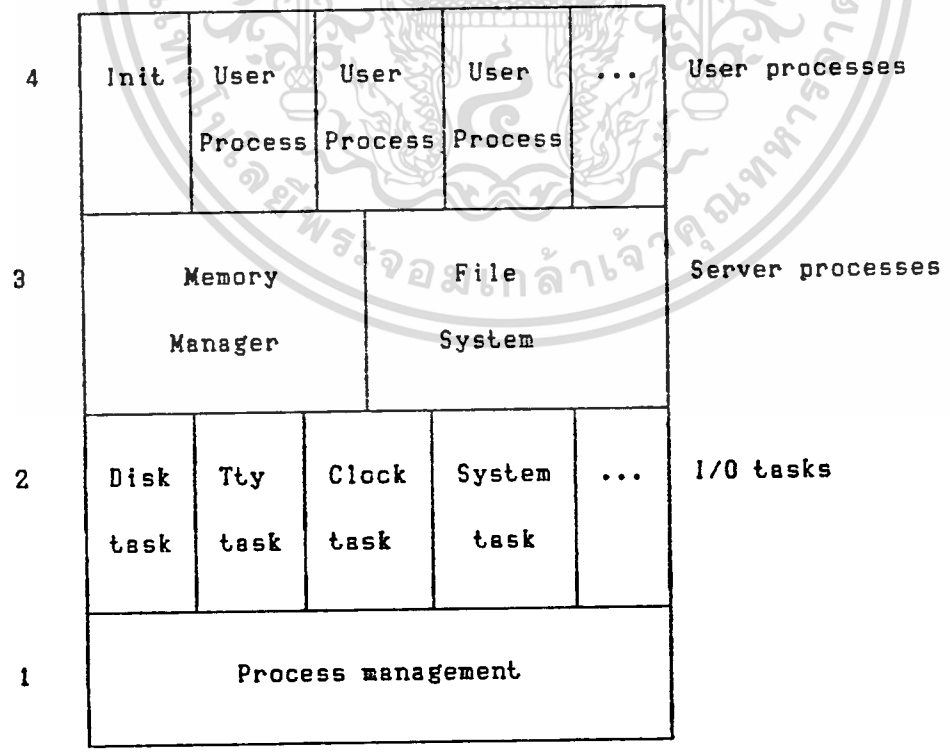
บทที่ 2

โครงสร้างและการทำงานของระบบปฏิบัติการมินิกซ์

โครงสร้างภายในระบบปฏิบัติการมินิกซ์

มินิกซ์ (MINIX) เป็นระบบปฏิบัติการที่เป็นสับเซตของยูนิกซ์ ทำงานบนเครื่องไอบีเอ็มพีซี/เอ็กซ์ที สามารถดำเนินงานหลายๆงานได้พร้อมๆกัน (multitasking) โดยตัวมินิกซ์จะประกอบไปด้วยหลายโปรเซสที่ติดต่อ (communicate) กัน ภายในแต่ละโปรเซสด้วยการติดต่อโปรเซส (interprocess communication) แบบพื้นฐานวิธีเดียวคือวิธีการรับส่งข่าวสาร (message passing)

โครงสร้างของมินิกซ์จะแบ่งออกเป็นระดับ(layer)ใหญ่ๆได้ 4 ระดับด้วยกัน แสดงไว้ดังรูป



รูป 2.1 โครงสร้างในระดับต่างๆของมินิกซ์

ในระดับต่ำสุดจะดักอินเตอร์รัพท์และแทรป (trap) ที่เกิดขึ้นและช่วยจัดการติดต่อข่าวสารของโปรเซสในระดับสูงกว่า โปรแกรมในระดับนี้จะทำหน้าที่หลัก 2 ประการ คืออย่างแรกจะทำการจับการขออินเตอร์รัพท์และแทรปที่เกิดขึ้น เก็บรีจิสเตอร์และจัดการกับสิ่งที่โปรแกรมที่ขออินเตอร์รัพท์ต้องการ อย่างที่สองคือดูแลการส่งข่าวสารของแต่ละโปรเซส เช่น ตรวจดูความถูกต้องของตำแหน่งปลายทาง แอดเดรสของบัฟเฟอร์ของข่าวสารทั้งตัวรับและตัวส่งในหน่วยความจำจริงและลอกข้อมูลจากผู้ส่งไปยังผู้รับ เป็นต้น

ในระดับสองจะประกอบไปด้วยโปรเซสอินพุทเอาท์พุท (I/O process) หนึ่งโปรเซสต่ออุปกรณ์หนึ่งชนิด โดยจะใช้คำว่าทาสค์ (task) แทนคำว่าโปรเซส เพื่อแยกความแตกต่างระหว่างโปรเซสอินพุทเอาท์พุทกับโปรเซสของผู้ใช้ (user process) นอกจากนี้ในระบบอื่นมักเรียกไอโอทาสค์นี้ว่าดีไวซ์ไดรฟ์เวอร์(devicedriver) ซึ่งสามารถใช้คำแทนกันได้ อุปกรณ์แต่ละตัวจะมีทาสค์ของตัวเองไม่ว่าจะเป็นดิสค์, เครื่องพิมพ์, เทอร์มินัล, คล็อค หรือแม้แต่ทาสค์ของระบบเอง

ทาสค์ทั้งหมดในระดับสอง รวมทั้งโปรแกรมในระดับหนึ่งจะถูกลิงค์รวมกันเป็นไบนารีไฟล์เดียวกันเป็นเคอร์นัล (kernel) เพื่อง่ายต่อการพอร์ตโปรแกรมไปยังเครื่องที่มีระดับการทำงานเพียงสองระดับคือระดับเคอร์นัลและระดับยูสเซอร์เท่านั้น

ในระดับสาม จะประกอบไปด้วยสองโปรเซสที่ให้บริการแก่โปรเซสของผู้ใช้ ได้แก่ ตัวจัดสรรหน่วยความจำ (memory manager หรือ MM) ซึ่งจะจัดการกับฟังก์ชันคอลล์ทุกตัวที่เกี่ยวข้องกับการจัดการหน่วยความจำ เช่น fork, exec และ brk นอกจาก MM แล้วยังมีโปรเซสระบบจัดการไฟล์ (file system หรือ FS) ซึ่งจะจัดการกับฟังก์ชันคอลล์ที่มีการติดต่อกับไฟล์ เช่น read mount และ chdir เป็นต้น

จะเห็นว่า จากหน้าที่ของระบบดำเนินงาน มีอยู่สองหลักใหญ่ คือ จัดสรรทรัพยากร ซึ่งเป็นหน้าที่ในระดับหนึ่งและสอง ในขณะที่การอำนวยความสะดวกในการใช้งานด้วยฟังก์ชันคอลล์จะเป็นหน้าที่ของโปรเซสในระดับสาม

และในระดับสี่จะประกอบไปด้วยโปรเซสทั้งหมดของผู้ใช้ เช่น เชลล์, อิดิตเตอร์, คอมไพเลอร์ และโปรแกรมที่ผู้ใช้สร้างขึ้นเอง จึงจัดได้ว่าในระดับนี้เป็นระดับสำหรับโปรเซสของโปรแกรมประยุกต์ใช้งาน

รายละเอียดการทำงานในส่วนเคอร์เนล

การจัดการโปรเซสในมินิกซ์

โปรเซสในมินิกซ์มีความสามารถในการสร้างโปรเซสย่อย(subprocess)ได้ ดังนั้น แต่ละโปรเซสสามารถสร้างโปรเซสย่อยได้เรื่อยๆ จนกลายเป็นโปรเซสที่เป็นโครงสร้างต้นไม้ โดยมีราก (root) ของต้นไม้อยู่ที่โปรเซส init ซึ่งถือว่าเป็นโปรเซสแรกของระบบ

ขั้นตอนการเข้าสู่มินิกซ์ จะเริ่มต้นหลังจากเปิดเครื่อง เมื่อเครื่องอ่านข้อมูลในเซกเตอร์และแทรคแรกของแผ่นดิสค์เข้าสู่หน่วยความจำแล้ว ก็จะกระโดดไปที่ตำแหน่งนั้น เพื่อที่จะทำงานตามโปรแกรมที่อยู่ในบูตสแตรป (bootstrap) ซึ่งจะทำการโหลดระบบปฏิบัติการทั้งหมดลงสู่หน่วยความจำและทำงาน หลังจากเคอร์นัล, ตัวจัดสรรหน่วยความจำ และระบบจัดการไฟล์ทำการติดตั้งและเซตค่าต่างๆเรียบร้อยแล้ว การควบคุมทั้งหมดก็จะไปสู่โปรเซส init

โปรเซส init ทำงานโดยเริ่มต้นจะอ่านไปที่ไฟล์ /etc/ttys เพื่อที่จะดูว่ามีเทอร์มินัลต่อรวมกันอยู่ในขณะนั้นเท่าใด (โดยปรกติจะมีเพียงคอนโซลเท่านั้นสำหรับการใช้เพียงเครื่องเดียว) และ จะทำการฟอร์ค (fork) โปรเซสย่อยตามจำนวนเทอร์มินัล และในแต่ละโปรเซสจะไปเรียกโปรแกรมใน /bin/login ทำงาน และคอยจนกว่าจะมีผู้ใช้เข้ามาในระบบ

หลังจากมีผู้ใช้เข้ามาในระบบเรียบร้อยแล้ว โปรแกรม login จะไปเรียกโปรแกรมเชลล์ซึ่งปรกติอยู่ที่ไฟล์ /bin/sh โดยการทำงานของเชลล์จะคอยคำสั่งที่ป้อนจากผู้ใช้ และฟอร์คโปรเซสใหม่ตามคำสั่งแต่ละบรรทัด จะเห็นได้ว่าเชลล์เป็นโปรเซสลูกของ init ในขณะที่โปรเซสของผู้ใช้จะเป็นโปรเซสหลาน ดังนั้น ทุกๆโปรเซสของระบบ จะเป็นส่วนหนึ่งของโปรเซสในโครงสร้างต้นไม้ที่มีรากเป็นโปรเซส init

สำหรับการจัดการโปรเซส (process management) จะมีฟังก์ชันคอลล์ที่สำคัญ 2 ตัวคือ fork และ exec โดย fork เป็นฟังก์ชันเดียวที่ใช้ในการสร้างโปรเซสใหม่ ในขณะที่ exec จะอนุญาตให้มีการสั่งทำงานโปรแกรมที่ต้องการ ส่วนรายละเอียดของแต่ละโปรเซสจะถูกเก็บไว้ในตารางโปรเซส (process table) โดยแต่ละเคอร์นัล,ส่วนจัดการหน่วยความจำ

ระบบจัดการไฟล์ จะมีตารางโปรเซสของตัวเอง เมื่อมีการสร้างโปรเซสใหม่ หรือโปรเซสจบการทำงานด้วยคำสั่ง exit หรือได้รับสัญญาณ (signal) แล้วส่วนจัดการหน่วยความจำจะทำการเปลี่ยนแปลงฟิวด์ต่างๆในตารางโปรเซส และจะส่งข่าวสารไปยังส่วนจัดการไฟล์และเคอร์นัล เพื่อให้โปรเซสนั้นๆเปลี่ยนแปลงข้อมูลในตารางโปรเซสของตัวเองต่อไป

โครงสร้างของตารางโปรเซส (Process table)

ในมินิกซ์นี้ ได้มีการจัดการโปรเซสต่าง ๆ ให้สามารถแบ่งเวลาใช้งานซีพียูได้ เพื่อให้มีการทำงานเป็นลักษณะของหลายๆ โปรเซสพร้อมๆกัน โดยมินิกซ์ได้เก็บรายละเอียดต่างๆและสถานะของโปรเซสแต่ละโปรเซสไว้ในโครงสร้างข้อมูลซึ่งในที่นี้จะเรียกว่า โปรเซสเทเบิล

โปรเซสเทเบิลนี้ ในตัวโปรแกรมจะเป็นอาเรย์ (array) ของโครงสร้างข้อมูลที่มีชื่อว่า proc (ไฟล์ proc.h) โดยจะมีจำนวนสล้อตทั้งหมดเท่ากับ NR_TASKS + NR_PROCS (นิยามไว้ในไฟล์ const.h และมีค่าเท่ากับ 8 และ 16 ตามลำดับ) โดยแต่ละสล้อตจะมีลักษณะโครงสร้างข้อมูลดังรูป 2.2

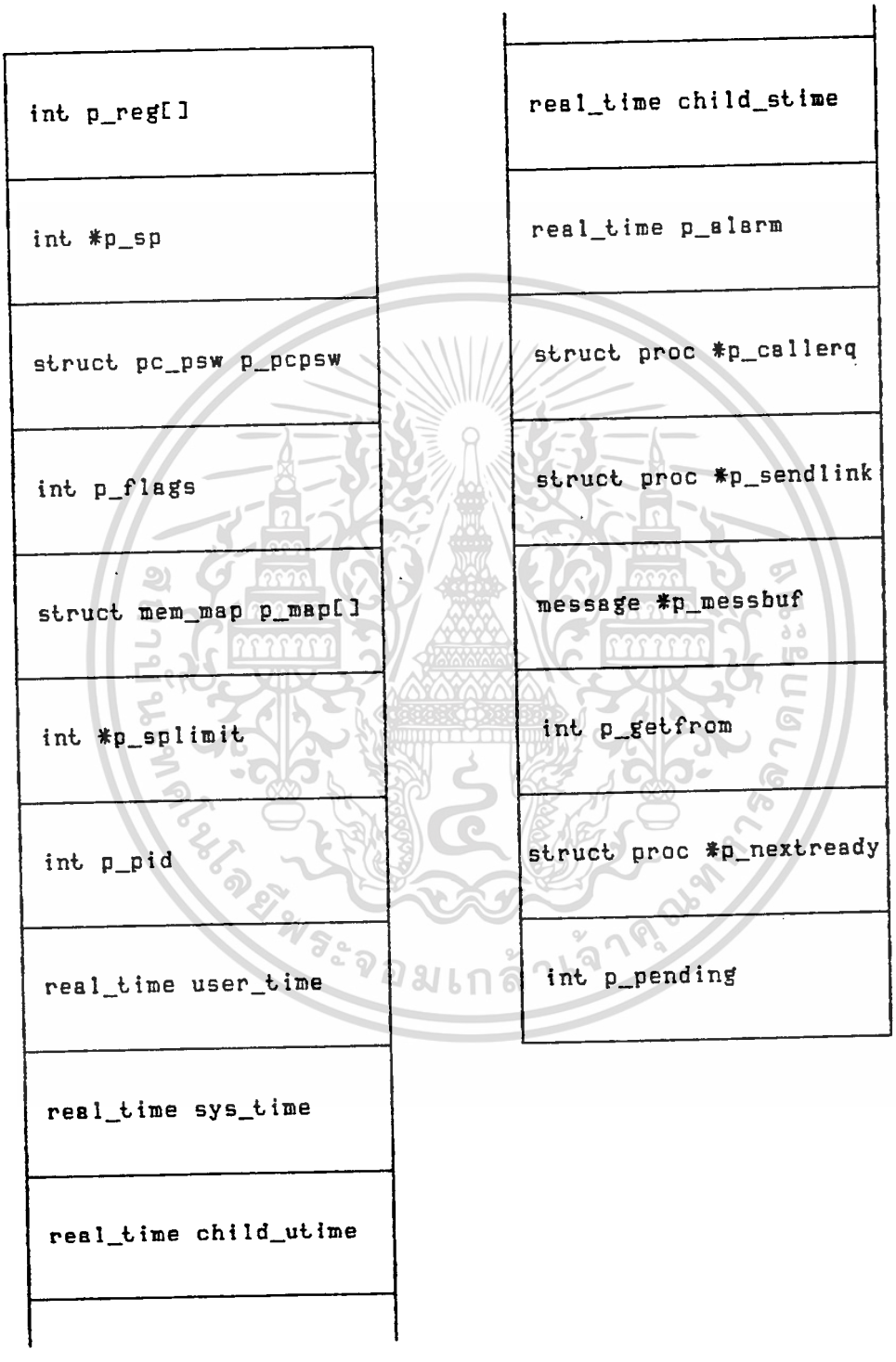
ความหมายของข้อมูลแต่ละตัวเป็นดังนี้ คือ

p_reg เป็นอาเรย์ของจำนวนเต็ม 16 บิต ใช้ในการเก็บค่าในรีจิสเตอร์ทั้งหมด ในช่วงการสวิทช์โปรเซส ทั้งหมด NR_REGS ตัว (ไฟล์ const.h มีค่าเท่ากับ 11) อันได้แก่ ax , bx, cx, dx, si, di, bp, es, ds, cs และ ss ตามลำดับ

p_sp เป็นที่เก็บค่าของสแต็กพอยน์เตอร์ (sp) ในช่วงการสวิทช์โปรเซส

p_pcpsw เป็นที่เก็บโปรแกรมเคาน์เตอร์ (pc), โค้ดเซ็กเมนต์ (cs) และสถานะของโปรเซส หรือแฟล็กนั่นเอง ตามลำดับ โดยโครงสร้างข้อมูล pc_psw กำหนดไว้ในไฟล์ type.h ดังรูป 2.3

struct proc



รูป 2.2 โครงสร้างของตารางโปรเซส

struct pc_psw

int (*pc)()	โปรแกรมเคาน์เตอร์
phys_clicks cs	โค้ดเซกเมนต์
unsigned psw	สถานะโปรแกรม ซึ่งก็คือ แฟลก

รูป 2.3 โครงสร้างข้อมูล pc_psw

เป็นโครงสร้างที่จะเก็บลงสแต็ก เมื่อเกิดอินเทอร์รัปต์ ดังนั้น โครงสร้างนี้จึงขึ้นอยู่กับฮาร์ดแวร์ของเครื่อง ซึ่งในที่นี้เป็นลักษณะของซีพียู 80xxx ของบริษัทอินเทล

p_flags เป็นตัวบอกสถานะของโปรเซสนั้น ๆ ก่อนที่จะถูกบล็อก โดยจะเป็นไปได้ทั้งหมด 4 สถานะ ได้แก่ P_SLOT_FREE หมายความว่าสล้อตนี้ ยังไม่ถูกใช้โดยโปรเซสใด ๆ เลย, NO_MAP หมายความว่าโปรเซสนั้นๆ ถูกสร้าง (FORK) แล้วแต่ยังไม่มีการแม็ปหน่วยความจำให้, SENDING หมายความว่าโปรเซสนั้น กำลังรอที่จะส่งข้อความอยู่ แต่โปรเซสเป้าหมายไม่ว่างที่จะทำการรับข้อความ และสถานะสุดท้าย คือ RECEIVING หมายความว่า โปรเซสนั้น ๆ กำลังรอข้อความอยู่ แต่ยังไม่มีโปรเซสใด ๆ ส่งข้อความมาเลย

p_map เป็นที่เก็บแอดเดรสของแต่ละส่วนของโปรเซส โดยจะประกอบด้วย 3 ส่วนได้แก่ T (text), D (data) และ S (stack) โดยแอดเดรสทั้ง 3 ส่วนจะเก็บในลักษณะของโครงสร้างข้อมูล struct mem_map (ไฟล์ type.h) คือจะเก็บแอดเดรสเสมือน (virtaul address), แอดเดรสจริง (physical address) และ ขนาดของหน่วยความจำส่วนนั้น ๆ

p_splimit เป็นขนาดของสแต็ก โดยเก็บเป็นตำแหน่งต่ำสุดที่เป็นไปได้ของสแต็ก

p_pid เป็นหมายเลขประจำโปรเซสนั้น ๆ (ไม่ใช่หมายเลขของอาเรย์)

user_time,sys_time เป็นที่เก็บว่าโปรเซสนั้นๆ ทำงานในโหมดของยูสเซอร์ (usermode) และ ระบบ (system mode) ไปนานเท่าใดแล้ว โดยจะเก็บเป็นจำนวนครั้ง (ticks) ของสัญญาณนาฬิกา โดยฟังก์ชั่นที่ทำหน้าที่นับเวลานี้ คือ ฟังก์ชั่น accounting ในไฟล์ clock.c

child_utime, child_stime ใช้ในลักษณะเดียวกับ user_time และ sys_time แต่เป็นระยะเวลาในการทำงานในทั้งสองโหมดของโปรเซสลูก ของโปรเซสนั้น ๆ อีกทีหนึ่ง

p_alarm เป็นระยะเวลาที่โปรเซสต้องการอยู่ในสถานะบล็อก และเมื่อครบระยะเวลาโปรเซส clock จะทำการส่งสัญญาณ (signal) มาบอกโปรเซสนั้น ๆ โดยฟังก์ชั่นที่ทำหน้าที่นับเวลานี้ คือ do_clocktick ในไฟล์ clock.c

p_callerq, p_sendlink ใช้ในการรับส่งข้อความซึ่งจะอธิบายในหัวข้อการส่งข้อความ

p_messbuf เป็นแอดเดรสของบัฟเฟอร์ของข้อความของแต่ละโปรเซส

p_getfrom ถ้าโปรเซสต้องการรับข้อความจากโปรเซสใด แต่โปรเซสนั้นยังไม่ทำการส่งข้อความมา โปรเซสที่รอ ก็จะถูกบล็อก โดยที่จะใส่หมายเลขของโปรเซสที่ต้องการให้ส่งข้อความมาให้ ไว้ที่ p_getfrom นี้เอง

p_nextready เป็นพอยน์เตอร์ที่ชี้ไปยังโปรเซสเทเบิลของโปรเซสต่อไปที่พร้อม

จะทำงานเป็นการเชื่อมต่อกันเป็นลิงค์ลิสต์ของโปรเซส

p_pending เป็นแฟล็กที่บอกว่ามีสัญญาณใดบ้าง ที่ถูกส่งมาให้โปรเซสนั้น ๆ ในขณะที่โปรเซสถูกบล็อกอยู่ โดยจะเป็นแฟล็กขนาด 16 บิท ซึ่งแต่ละบิทจะแทนสัญญาณแต่สัญญาณแต่ละสัญญาณ (สัญญาณในมินิกซ์ มีทั้งหมด 16 สัญญาณ แสดงในไฟล์ signal.h) อนึ่ง ในการอ้างอินเด็กซ์ของอาเรย์ของ proc ถ้าเป็นโปรเซสของระบบ (tasks) จะใช้เลขติดลบ ซึ่งจะดูได้จากไฟล์ com.h ในขณะที่ส่วนจัดการหน่วยความจำ (memory management) และส่วนระบบไฟล์ (file system) จะใช้หมายเลข 0 และ 1 ตามลำดับ ส่วนโปรเซสของผู้ใช้จะเริ่มที่หมายเลข 2 เป็นต้นไป โดยหมายเลข 2 จะเป็นของโปรเซสเริ่มต้น (init)ทั้งหมดนี้จะดูได้จากไฟล์ const.h

ในภาษาซี อาเรย์จะมีอินเด็กซ์เป็นเลขติดลบไม่ได้ ดังนั้นการที่จะอ้างถึง proc จะต้องอ้างผ่านมาโคร proc_addr ซึ่งกำหนดไว้ในไฟล์ proc.h ซึ่งจะทำหน้าที่แปลงหมายเลขอินเด็กซ์ให้ถูกต้อง

การติดต่อระหว่างโปรเซสในมินิกซ์

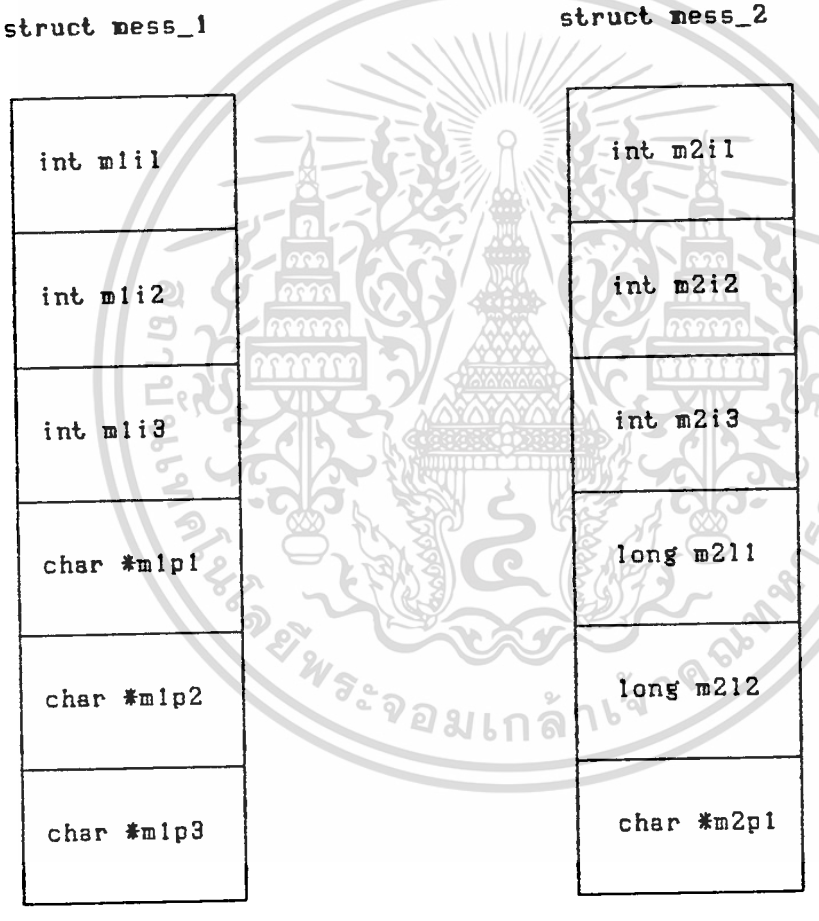
การติดต่อระหว่างโปรเซสจะใช้วิธีการส่งข่าวสารที่มีขนาดคงที่ (fixed size) โดยขนาดขึ้นอยู่กับขนาดของข่าวสารแต่ละชนิด ในมินิกซ์จะแบ่งข่าวสารเป็น 6 ชนิด โดยแต่ละชนิดจะมีขนาดคงที่และเก็บข้อมูลที่มีชนิดแน่นอน

เมื่อโปรเซส (รวมทั้งทาสค์) ส่งข่าวสารไปยังโปรเซสที่ยังไม่ได้คอยข่าวสารอยู่ โปรเซสที่ส่งจะถูกบล็อคจนกระทั่งโปรเซสปลายทางรับข่าวสารด้วยคำสั่ง receive การรับส่งข่าวสารของโปรเซสผู้ใช้มีข้อจำกัดคือไม่สามารถรับส่งข่าวสารโดยตรงกับไอโอทาสค์ เพราะขัดกับข้อกำหนดของระบบ

โครงสร้างของข่าวสาร

ในการติดต่อระหว่างโปรเซสของมินิกซ์ ใช้การส่งข่าวสาร ซึ่งข่าวสารในมินิกซ์มีทั้งหมด 6 ประเภท ตามรูปแบบของพารามิเตอร์ที่ต้องการส่ง เช่น ถ้าต้องการส่งพารามิเตอร์ เป็นเลขจำนวนเต็ม 3 ตัว และพอยน์เตอร์ 3 ตัว ก็ใช้โครงสร้าง mess_1 เป็นต้น

สำหรับโครงสร้างของส่วนข่าวสารในการติดต่อกันระหว่างโปรเซสทั้ง 6 ประเภท แสดงไว้ในรูป 2.4



รูป 2.4 โครงสร้างของส่วนข่าวสาร

struct mess_3

int m3i1
int m3i2
char *m3p1
char m3ca1[]

struct mess_4

long m411
long m412
long m413
long m414

struct mess_5

char m5c1
char m5c2
int m5i1
int m5i2
long m511
long m512
long m513

struct mess_6

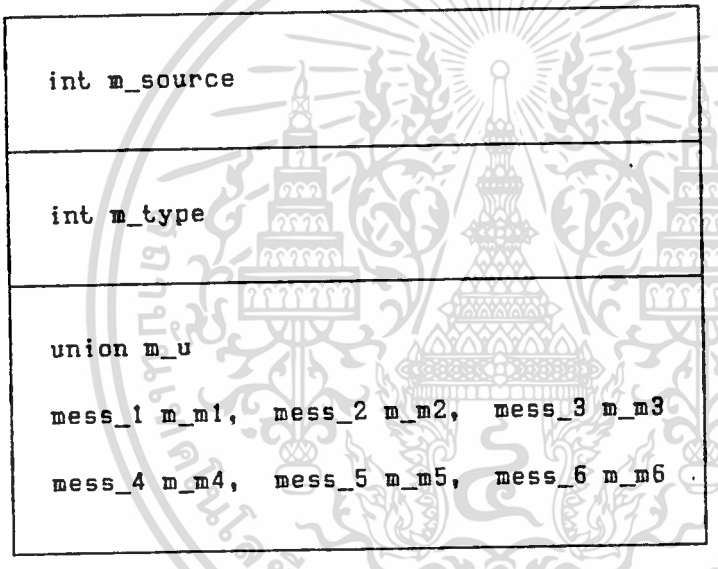
int m6i1
int m6i2
int m6i3
long m611
int(*m6f1)()

รูปที่ 2.4 (ต่อ)

โครงสร้างของข่าวสารจริงๆที่ใช้ในการรับส่ง จะมี 3 ส่วน ดังรูป 2.5 ได้แก่

- m_source เป็นส่วนบอกว่าโปรเซสใด เป็นผู้ส่ง
- m_type เป็นส่วนบอกว่าเป็นข้อความชนิดใด
- m_u เป็นส่วนข่าวสาร ซึ่งเป็นได้ 6 ประเภท ดังที่กล่าวไปแล้ว

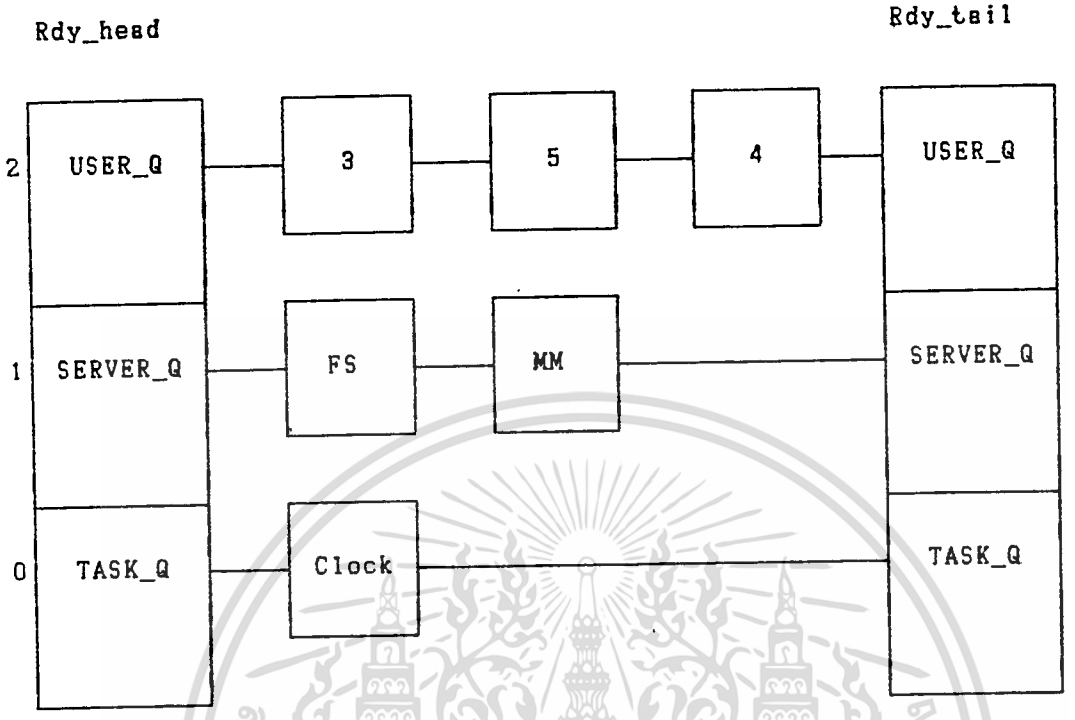
```
struct message
```



รูป 2.5 โครงสร้างของข่าวสารที่ใช้ในการติดต่อกันระหว่างโปรเซส

การจัดสรรเวลาของโปรเซสในมินิกซ์

ส่วนจัดสรรเวลาของโปรเซสในมินิกซ์ ใช้วิธีเข้าคิวหลายระดับ (multilevel queueing system) โดยแบ่งเป็นสามระดับตามระดับ 2,3,4 ในรูปที่ 2.6 ในแต่ละระดับจะจัดสรรเวลาด้วยวิธีของราวด์โรบิน (round robin) โดยทาสค์จะมีระดับความสำคัญ (priority) สูงสุดรองลงมาคือ ส่วนจัดสรรหน่วยความจำ, ส่วนจัดการระบบไฟล์ และโปรเซสของผู้ใช้ตามลำดับ



รูป 2.6 ตัวจัดสรรเวลาจัดคิวแก่โปรเซสแต่ละระดับ

ในการเลือกโปรเซสที่จะนำมาทำงาน ส่วนจัดสรรจะตรวจดูว่ามีทาสค์ใดพร้อมที่จะทำงานบ้าง ถ้ามีแล้วทาสค์แรกที่อยู่ในคิวจะเลือกมาทำงาน ถ้าไม่มีทาส์คที่พร้อม MM, FS จะเป็นส่วนถัดไปที่จะพิจารณา มิฉะนั้นโปรเซสของผู้ใช้จะถูกเลือก แต่ถ้าไม่มีโปรเซสใดพร้อมที่จะทำงาน ระบบจะวนลูปคอยจนกว่าจะมีการอินเตอร์รัพท์ครั้งต่อไป

ในแต่ละสัญญาณนาฬิกาที่มากระตุ้นระบบจะมีการตรวจเช็คว่าโปรเซสนั้นเป็นโปรเซสของผู้ใช้ที่ทำงานมานานกว่า 100 มิลลิวินาทีหรือไม่ ถ้าใช่แล้วส่วนจัดสรรเวลาจะเลือกโปรเซสถัดไปมาทำงานต่อและจะนำโปรเซสในขณะนั้นไปเก็บไว้ในส่วนท้ายของคิวแทน ซึ่งเรียกการตรวจเช็คเวลาการทำงานของโปรเซสจะเรียกว่า การขัดจังหวะโปรเซส (process preemption) ซึ่งโปรเซสในระดับอื่น (เคอร์นัล, MM, FS) จะไม่มีการขัดจังหวะการทำงานไม่ว่าจะทำงานมานานเท่าใด

ระบบการอินเทอร์รัปต์ในมินิกซ์

โดยปรกติโปรเซสที่กำลังทำงานอยู่ อาจจะถูกขัดจังหวะการทำงานด้วยสัญญาณอินเทอร์รัพท์ ไม่ว่าจะเป็นการขัดจังหวะจากสัญญาณนาฬิกา จากเทอร์มินัล หรือจากสัญญาณอื่น ๆ ดังนั้นหน้าที่ส่วนหนึ่งของเคอร์นัล ซึ่งจัดได้ว่าเป็นส่วนที่อยู่ในระดับต่ำที่สุดของระบบที่สำคัญอีกประการหนึ่ง ก็คือการซ่อนการขัดจังหวะต่างๆเหล่านี้ โดยการเปลี่ยนสัญญาณเป็นข่าวสารแทน

การทำงานของทาสค์ต่าง ๆ ในมินิกซ์

มินิกซ์ ประกอบด้วยทาสค์ต่าง ๆ และ มีรายละเอียดการทำงาน ดังนี้

(1) แรมดิสค์ทาสค์

RAM Disk Drivers ของ MINIX จะแบ่งออกเป็น 4 ชนิด คือ

0 : /dev/ram

1 : /dev/mem

2 : /dev/kmem

3 : /dev/null

/dev/ram คือไดร์ฟเวอร์ของแรมดิสค์จริงๆ ที่ถูกสร้างขึ้นจาก file system โดยเริ่มจากการบูตด้วยแผ่น MINIX

/dev/mem คือไดร์ฟเวอร์ของการเข้าหาหน่วยความจำแบบ physical ซึ่งจะเริ่มต้นที่ absolute 0

/dev/kmem จะทำหน้าที่เหมือน /dev/mem แต่การอ้างถึงหน่วยความจำจะอ้างที่ตำแหน่งphysical แอดเดรส 0x600 หรือ 1536 เป็นตำแหน่ง 0 ใน kernel memory

/dev/null เป็นไดร์ฟเวอร์ที่จะนำข้อมูลทิ้งไปหรือไม่ปรากฏให้เห็น เช่นการเรียก a.out >/dev/null ผลลัพธ์ของ a.out จะไม่ปรากฏผลลัพธ์ออกมาหรือถูกทำให้เป็นค่า NULL

(2) ฟล้อปปี้ดิสค์ทาสค์

ฟล้อปปี้ทาสก์ เป็นโปรเซสของระบบ ที่ทำหน้าที่จัดการกับการติดต่อกับฟล้อปปี้ดิสก์ โดยที่ตัวโปรแกรมจะอยู่ในไฟล์ floppy.c ในตัวโปรแกรมจะมีตัวแปรที่สำคัญคือ floppy ซึ่งเป็นอาเรย์ขนาด 2 ตัว (ตัวที่ 0 และตัวที่ 1 ใช้เก็บข้อมูลของไดรฟ์ A และ B ตามลำดับ) ของโครงสร้าง struct floppy ซึ่งมีโครงสร้างดังรูป 2.7 คือ

```
int fl_opcode;
int fl_curcyl;
int fl_procnr;
int fl_drive;
int fl_cylinder;
int fl_sector;
int fl_head;
int fl_count;
vir_bytes fl_address;
char fl_results[MAX_RESULT];
char fl_calibration;
```

รูป 2.7 โครงสร้างข้อมูลของ floppy

โดยข้อมูลแต่ละตัวในโครงสร้าง จะมีรายละเอียดดังนี้

fl_opcode ใช้เก็บสถานะว่าคำสั่งที่กำลังทำเป็นการอ่าน หรือเขียน

fl_curcyl ใช้เก็บว่าปัจจุบันหัวอ่านอยู่ที่ไซลินเดอร์ใด

fl_procnr ใช้เก็บว่าโปรเซสใด เป็นโปรเซสที่ต้องการ อ่าน/เขียนดิสก์ โดยหมายเลขที่เก็บคือ หมายเลขสล้อตของโปรเซสเทเบิล (ไม่ใช่เบอร์โปรเซส)

fl_drive ใช้เก็บว่าเป็นคำสั่งอ่าน/เขียน ของไดรฟ์ใด

fl_cylinder, fl_sector, fl_head ใช้เก็บว่า เป้าหมายที่ต้องการอ่าน/เขียน อยู่ที่ไซลินเดอร์ใด, เซ็กเตอร์ใด และด้านใดของแผ่นดิสก์ ตามลำดับ

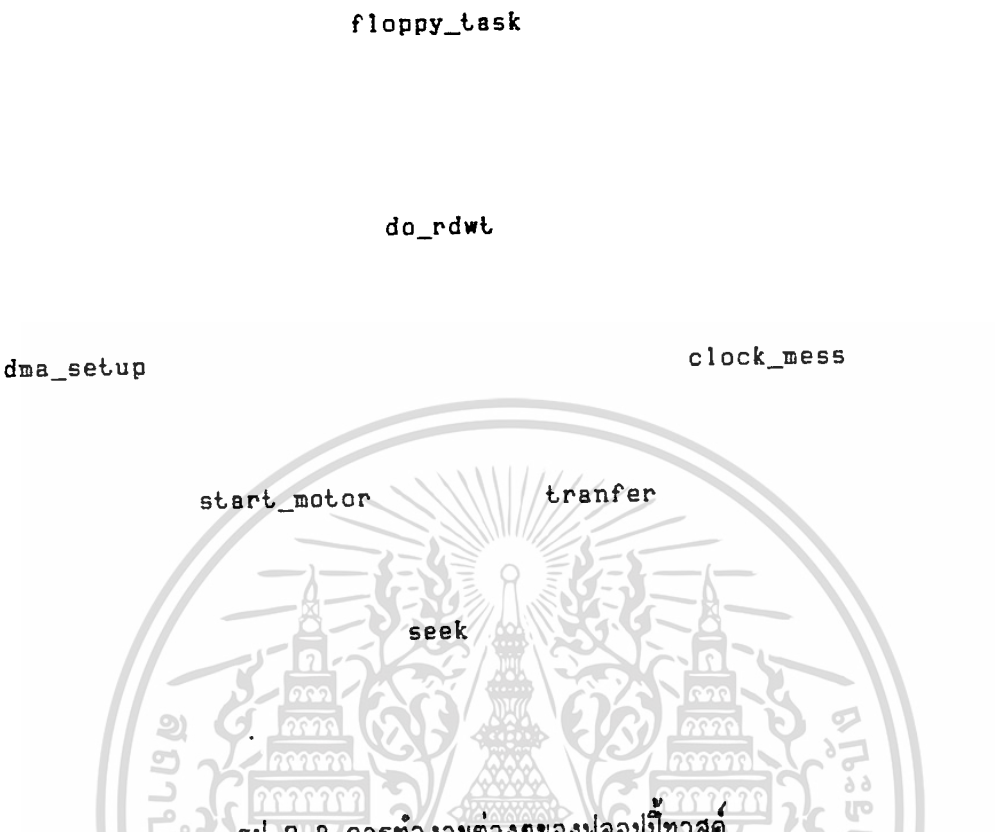
fl_count ใช้เก็บจำนวนไบต์ที่ต้องการอ่าน/เขียน

fl_address ใช้เก็บแอดเดรสของข้อมูลที่ต้องการเขียนลงแผ่นดิสก์ในกรณีที่ทำการเขียน หรือแอดเดรสที่ต้องการนำข้อมูลจากแผ่นดิสก์มาใส่ ในกรณีที่ทำการอ่าน

fl_results ใช้เก็บผลลัพธ์ ที่ได้จากการสั่งให้ดิสก์คอนโทรลเลอร์ ทำงาน โดยในแต่ละกรณีจะให้ผลลัพธ์คืนมาจำนวนไม่เท่ากัน แต่จะมากที่สุดเท่ากับ MAX_RESULTS

fl_calibration ใช้เก็บว่าไดรฟ์นั้น ๆ ต้องการการเลื่อนหัวอ่าน/เขียน กลับไปที่ไซลินเดอร์ 0 หรือไม่ ซึ่งจะใช้ในกรณีที่เกิดความผิดพลาดของการเลื่อนหัวอ่าน/เขียนไป-มา

การทำงานฟังก์ชันต่าง ๆ ของฟล้อปปี้ทาสก์ มีลักษณะเป็นดังรูป 2.8



รูป 2.8 การทำงานต่างๆของฟลอปปี้ทาสค์

(3) ซิสเต็มทาสค์

ทาสก์นี้ ต่างกับทาสก์อื่น ๆ คือ ไม่ได้ทำหน้าที่เป็นตัวติดต่อกับอุปกรณ์ภายนอกต่าง ๆ เหมือนกับทาสก์อื่น ๆ เช่น ทีทีวายทาสก์ที่ทำหน้าที่ติดต่อกับเทอร์มินัล หรือฟล้อปปี้ทาสก์ซึ่งทำหน้าติดต่อกับฟล้อปปี้ดิสก์ หากแต่ซิสเต็มทาสก์จะทำหน้าที่เป็นตัวกลางระหว่างส่วนเคอร์เนล(kernel) กับส่วนระบบไฟล์ (file system) และ ส่วนจัดการหน่วยความจำ (memory management)

สาเหตุที่ต้องมีซิสเต็มทาสก์ ก็เพราะว่า มินิกซ์ถูกเขียนขึ้นมาโดยให้มีความเป็นโครงสร้างสูง โดยจะแบ่งส่วนเคอร์เนล ,ส่วนระบบไฟล์ และส่วนจัดการหน่วยความจำ ออกจากกันโดยเด็ดขาด คือ จะเปรียบเสมือนโปรแกรมสามโปรแกรม ที่ทำงานอยู่ในหน่วยความจำพร้อม ๆ กัน โดยการติดต่อกัน จะใช้ระบบการส่งข่าวสาร ดังที่ได้เคยกล่าวมาแล้ว ดังนั้นซิสเต็มทาสก์จึงเปรียบเสมือนจุดติดต่อของส่วนเคอร์เนลนั่นเอง

ข่าวสารที่สามารถส่งมาที่ซิสเต็มทาสก์ได้ มีอยู่ 9 ชนิด ได้แก่

SYS_FORK ทำหน้าที่บอกส่วนเคอร์เนลว่ามีโปรเซสได้ถูกจำลอง (FORK) ขึ้นใหม่แล้ว โดยส่วนจัดการหน่วยความจำจะบอกให้ส่วนเคอร์เนล ทำการเปลี่ยนแปลงโปรเซสเทเบิลของส่วนเคอร์เนล (ทั้งสามส่วน คือเคอร์เนล ,ระบบไฟล์ และส่วนจัดการหน่วยความจำ ต่างก็มีโปรเซสเทเบิลของตัวเองทั้งสิ้น)

SYS_NEWMAP ทำหน้าที่บอกส่วนเคอร์เนลเกี่ยวกับตำแหน่งในหน่วยความจำ ของโปรเซส โดยส่วนจัดการหน่วยความจำจะบอกให้ส่วนเคอร์เนล ทำการเปลี่ยนแปลงโปรเซสเทเบิลของส่วนเคอร์เนล

SYS_EXEC ทำหน้าที่เซ็ตค่าของสแต็กพอยน์เตอร์ ของโปรเซสที่ถูกสร้างขึ้นใหม่ (EXEC) โดยส่วนจัดการหน่วยความจำจะเป็นส่วนที่บอกเคอร์เนล

SYS_XIT ส่วนจัดการหน่วยความจำจะเป็นส่วนบอกเคอร์เนลว่า มีโปรเซสที่สิ้นสุดการทำงานแล้ว (EXIT)

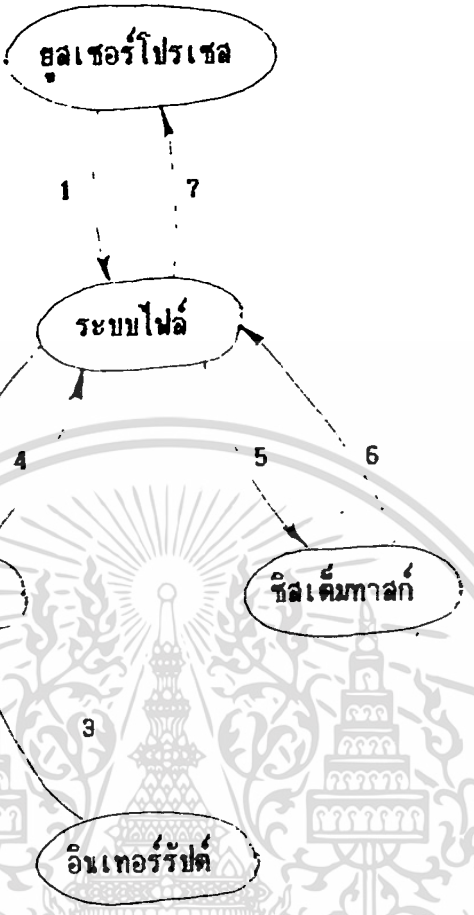
SYS_GETSP ส่วนจัดการหน่วยความจำ ต้องการทราบค่าของสแต็กพอยน์เตอร์ของโปรเซสใด ๆ

SYS_TIMES ระบบไฟล์ต้องการทราบระยะเวลาทั้งหมดที่โปรเซสใด ๆ ได้ทำงานไปแล้ว ทั้งในเคอร์เนลโหมด และยูสเซอร์โหมด

SYS_ABORT ระบบไฟล์ หรือ ส่วนจัดการหน่วยความจำ ไม่สามารถทำงานต่อได้ ดั้งนั้นจึงบอกให้เคอร์เนลหยุดการทำงาน

SYS_SIG ทำหน้าที่ส่งสัญญาณ (signal) ไปที่โปรเซสใด ๆ โดยส่วนจัดการหน่วยความจำ จะเป็นส่วนจัดการเกี่ยวกับสัญญาณ ซึ่งจะได้กล่าวถึงภายหลัง

SYS_COPY ทำหน้าที่ในการก๊อปปี้ข้อมูลระหว่าง ยูสเซอร์โปรเซส และระบบไฟล์ หรือ ยูสเซอร์โปรเซส และส่วนจัดการหน่วยความจำ ตัวอย่างการทำงานจะเป็นดังรูป 2.9



รูป 2.9 แสดงตัวอย่างการทำงานในการอ่านข้อมูลจากดิสค์

ในรูปนี้ ยูสเซอร์โปรเซสต้องการอ่านข้อมูลที่อยู่ในแผ่นดิสค์ ยูสเซอร์โปรเซสก็จะทำการส่งข่าวสาร (1) บอกระบบไฟล์ ระบบไฟล์ก็จะทำการตรวจดูว่าข้อมูลที่ต้องการอยู่ในแคชหรือไม่ ถ้าไม่ ก็จะทำการส่งข่าวสาร (2) เพื่อบอกดิสก์ทาสก์ ให้ทำการอ่านข้อมูลจากแผ่นดิสก์ โดยเมื่อดิสก์คอนโทรลเลอร์อ่านข้อมูลจากแผ่นได้แล้ว ก็จะส่งอินเทอร์รัปต์ในรูปของข่าวสาร (3) บอกดิสก์ทาสก์ว่าได้อ่านข้อมูลเรียบร้อยแล้ว ดิสก์ทาสก์ก็จะส่งข่าวสาร (4) บอกระบบไฟล์ว่าอ่านข้อมูลมาใส่แคชเรียบร้อยแล้ว ระบบไฟล์ก็จะทำการส่งข่าวสาร (5) บอกให้ซิสเต็มทาสก์ทำการก๊อปปี้ข้อมูลจากแคชไปที่ยูสเซอร์โปรเซส เมื่อก๊อปปี้เสร็จ ซิสเต็มทาสก์ก็จะส่งข่าวสาร (6) บอกระบบไฟล์ ระบบไฟล์ก็จะส่งข่าวสาร (7) บอกยูสเซอร์โปรเซส

ส่วนในกรณีที่ข้อมูลอยู่ในแคชของระบบไฟล์อยู่แล้ว ข่าวสาร (2), (3) และ (4) ก็ไม่ต้องใช้ ดังนั้นในการอ่านข้อมูลจากดิสก์นั้น กรณีที่ดีที่สุด ต้องใช้การส่งข่าวสาร 4 ครั้ง คือ

(1) -> (5) -> (6) -> (7) ส่วนในกรณีที่เลวที่สุด ต้องใช้การส่งข่าวสารถึง 7 ครั้ง คือ (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) ดังนั้น จะเห็นได้ว่าจะเสียเวลามาก ซึ่งเป็นข้อเสียของโปรแกรมที่มีความเป็นโครงสร้างมาก ๆ นั่นเอง หากไม่สนใจความเป็นโครงสร้างของโปรแกรมแล้ว จะสามารถปรับปรุงความเร็วของมินิกซ์ได้ในส่วนนี้เอง

ข่าวสารทั้ง 9 ชนิดนี้ เนื่องจากต้องการความสะดวก ในซอร์สโค้ดจึงได้เขียนเป็นการเรียกไลบรารี โดยจะใช้ชื่อเป็นชื่อเดียวกับชนิดข่าวสาร แต่จะใช้ตัวอักษรตัวเล็กทั้งหมด เช่น ไลบรารี sys_sig จะเป็นการส่งข่าวสารชนิด SYS_SIG ไปที่ซิสเต็มทาสก์ ดังนั้น ถ้าเห็นในซอร์สโค้ดเป็น sys_xxx ก็หมายความว่า เป็นการส่งข่าวสารไปที่ซิสเต็มทาสก์

<u>รายละเอียดการทำงานในส่วนการจัดการหน่วยความจำ</u>

การจัดการกับหน่วยความจำ (Memory Management) ในมินิกซ์เป็นแบบง่ายๆ คือไม่ได้วิธีทั้ง pageing หรือ swapping การดูแลหน่วยความจำจะใช้ลิสต์ของโฮลที่เรียงกันตามตำแหน่งแอดเดรสในหน่วยความจำ โดยการค้นหาบล็อคข้อมูลในโฮลจะใช้วิธี first fit คือหาโฮลแรกในลิสต์ที่ใหญ่พอกับที่ต้องการจอง และเมื่อมีโปรเซสอยู่ในหน่วยความจำแล้วจะไม่มีการเคลื่อนย้ายหรือไปเก็บที่หน่วยความจำส่วนอื่นๆอีก ซึ่งวิธีที่ใช้นี้เหมาะสมกับการใช้ระบบดำเนินการนี้ในเครื่องไมโครคอมพิวเตอร์ที่ใช้ 8088 เป็นไมโครโปรเซสเซอร์ ซึ่งเป็นชิพที่ไม่มีการสนับสนุนการจัดการกับหน่วยความจำทางฮาร์ดแวร์ และเป็นเครื่องคอมพิวเตอร์ส่วนบุคคล (personnel computer) ที่ไม่เน้นการใช้งานในระบบขนาดใหญ่ที่มีการใช้งานร่วมกันกับโปรเซสจำนวนมาก

เนื่องจากการใช้งานมินิกซ์บนเครื่องพีซี จะมีจำนวนโปรเซสที่ทำงานอยู่ในระบบโดยเฉลี่ยไม่มาก ดังนั้นจำนวนหน่วยความจำจริงเพียงพอที่จะเก็บโปรเซสได้พร้อมๆกันทั้งหมด อีกทั้งการใช้งานมินิกซ์บนเครื่องพีซีนั้นไม่จำเป็นต้องมีฮาร์ดดิสค์จึงไม่เหมาะแก่การ swap โปรเซสลงในหน่วยความจำสำรอง

การจัดการกับข่าวสารในส่วนจัดการกับหน่วยความจำ

ส่วนจัดการกับหน่วยความจำ จะเป็นโปรเซสที่ติดต่อกับโปรเซสอื่นๆผ่านข่าวสาร เช่นเดียวกันกับส่วนอื่นๆในมินิกซ์ โดยหลังจากการตั้งค่าเริ่มต้นแล้ว ส่วนจัดการกับหน่วยความจำจะเข้าสู่ลูปที่คอยวนรับข่าวสารและทำงานตามข่าวสารที่ได้รับ แล้วจึงตอบโดยการสร้างข่าวสารส่งกลับไปยังผู้เรียกใช้ โดยมีชนิดการบริการข่าวสารดังตาราง 2.1

ชนิดของข่าวสาร	อินพุทพารามิเตอร์	ค่าตอบกลับ
FORK	ไม่มี	pid ของโปรเซสลูก
EXIT	สถานะการออกจากโปรแกรม	ไม่มีการตอบกลับ
WAIT	ไม่มี	รหัสสถานะ
BRK	ขนาดค่าใหม่	ขนาดค่าใหม่
EXEC	พอยน์เตอร์ไปที่สแต็ก	ไม่มีการตอบกลับ
SIGNAL	หมายเลขซิกแนลและฟังก์ชัน	ฟังก์ชันเดิม
KILL	pid ของโปรเซสและซิกแนล	0 ถ้าสำเร็จ
ALARM	จำนวนวินาทีที่คอย	เวลาที่คอย
PAUSE	ไม่มี	0 ถ้าสำเร็จ
GETPID	ไม่มี	pid ของโปรเซส
GETUID	ไม่มี	uid ของผู้เรียกใช้
GETGID	ไม่มี	gid ของผู้เรียกใช้
SETUID	uid ตัวใหม่	0 ถ้าเซตสำเร็จ
SETGID	gid ตัวใหม่	0 ถ้าเซตสำเร็จ
KSIG	สล็อตโปรเซสและซิกแนล	ไม่มีการตอบกลับ
BRK2	ขนาดตั้งต้นและขนาดรวมทั้งหมด	0 ถ้าสำเร็จ

ตาราง 2.2 แสดงบริการที่ส่วนจัดการหน่วยความจำสนับสนุน

การจัดการสัญญาณ

สัญญาณจะเกิดขึ้นได้โดย 2 วิธี ได้แก่ ซิสเต็มคอล KILL และสัญญาณที่เกิดจากส่วนเคอร์เนล โดยสัญญาณส่วนที่เกิดจากเคอร์เนล ในมินิกซ์เวอร์ชั่นปัจจุบันนี้ ได้แก่ SIGINT, SIGQUIT และ SIGALRM และโปรเซสที่จะสามารถส่งสัญญาณให้กันได้ ก็ต่อเมื่อ เป็นโปรเซสที่มี ยูไอดี (UID) เดียวกัน และเริ่มทำงานที่เทอร์มินัลเดียวกัน และโปรเซสนั้น ๆ ต้องไม่อยู่ในสถานะไม่สนใจสัญญาณ (ignore)

ในการส่งสัญญาณ จะเริ่มจากการที่ส่วนจัดการหน่วยความจำ จะส่งข่าวสาร บอกให้ซิสเต็มทาสก์ ทำการส่งค่า 4 เวิร์ด ลงสแต็ก ตามที่ได้กล่าวไปแล้วในฟังก์ชัน do_sig ซึ่งอยู่ในซิสเต็มทาสก์ เมื่อโปรเซสได้รับสัญญาณ ก็จะกระโดดไปทำงานที่ซิสเต็มรูทีนซึ่งจัดการเกี่ยวกับสัญญาณ โดยรูทีนนี้จะเก็บอยู่ในตารางโลคอลและสามารถเปลี่ยนแปลงได้โดยซิสเต็มคอล SIGNAL

การทำงานของรูทีนนี้ จะเริ่มด้วยการเซฟค่าของรีจิสเตอร์ต่าง ๆ ลงสแต็ก และใช้หมายเลขของสัญญาณที่อยู่ในสแต็ก เป็นอินเด็กซ์ในตารางฟังก์ชันภายใน เพื่อไปเรียกฟังก์ชันนั้น ๆ และหลังจากทำงานในฟังก์ชันนั้น ๆ เสร็จ ก็จะทำการเซ็ตค่าของรีจิสเตอร์ต่าง ๆ ที่ได้เซฟไว้ให้มีค่าเหมือนเดิม แล้วจะทำคำสั่งรีเทินจากอินเทอร์รัปต์ (return from interrupt) เพื่อจะทำงานต่อจากที่เดิมก่อนที่จะได้รับสัญญาณ

ถ้ามีการส่งสัญญาณไปที่โปรเซสใด ที่ยังไม่ได้ถูกอนุญาติให้รับสัญญาณได้ ส่วนจัดการหน่วยความจำก็จะทำการทำลาย (kill) โปรเซสนั้น ๆ

สำหรับโปรเซสที่กำลังรออินพุทจากเทอร์มินัล และมีสัญญาณถูกส่งมาให้ หลังจากจัดการกับสัญญาณเรียบร้อยแล้ว ซิสเต็มคอลจะรีเทิรน์ค่า EINTR เพื่อบอกให้ยูสเซอร์โปรเซสรู้ว่าซิสเต็มคอลสิ้นสุดการทำงานเพราะสัญญาณ ดังนั้นยูสเซอร์โปรเซสสามารถตัดสินใจได้ว่า จะทำการรออินพุทต่อ หรือไม่

สำหรับโปรเซสที่ต้องการจะรับสัญญาณ แต่มีเนื้อที่สแต็กไม่พอ ก็จะไม่ถูกส่งสัญญาณให้

การจองหน่วยความจำ

หน่วยความจำที่ว่างอยู่จะถูกเก็บอยู่ในโฮลลิสต์ การจองหน่วยความจำจะกระทำกับโฮลลิสต์นี้ โดยจะจองโดยใช้วิธี first fit โดยมีโครงสร้างข้อมูลของโฮลลิสต์ดังนี้

```
struct hole {
    phys_clicks h_base;          /* หมายเลขคลิกเริ่มต้น */
    phys_clicks h_len;           /* ขนาดของโฮล */
    struct hole *h_next;         /* พอยน์เตอร์ชี้ไปที่โฮลถัดไป */
} hole[NR_HOLES];
```

โดย NR_HOLES มีค่าเท่ากับ 128 และมีพอยน์เตอร์สองตัวคือ hole_head ชี้ไปที่โฮลแรก และ free_slots ชี้ไปที่สล็อตของโฮลที่ยังไม่ได้นำมาใช้ ทั้ง hole_head และ free_slots จะชี้ไปยังโฮลลิสต์ซึ่งเป็นลิงค์ลิสต์ทางเดียว (sigly linked list) โดยการจองหน่วยความจำจะไปค้นยังลิงค์ลิสต์ที่ hole_head ชี้อยู่ และเมื่อโฮลใดๆถูกใช้จนหมดก็จะนำไปเก็บไว้ในลิงค์ลิสต์ที่ชี้โดย free_slots เพื่อจะได้นำไปใช้งานต่อไป

<u>รายละเอียดการทำงานในระบบไฟล์</u>

ระบบไฟล์ในมินิกซ์ มีหน้าที่ในการจองและคืนเนื้อที่ว่างในหน่วยความจำสำหรับไฟล์, ติดตามและบันทึกการใช้บล็อคและเนื้อที่ว่างของดิสค์ , ป้องกันการเข้าถึงไฟล์สำหรับผู้ไม่มีสิทธิ์ และอื่นๆ สำหรับระบบไฟล์ของมินิกซ์จัดเป็นโปรเซสหนึ่งเช่นเดียวกับโปรเซสของผู้ใช้อื่นๆ เมื่อโปรเซสใดต้องการอ่านหรือเขียนไฟล์ โปรเซสเหล่านั้นก็จะส่งข่าวสารแสดงความต้องการมายังระบบไฟล์ เมื่อระบบไฟล์ทำงานเสร็จแล้วก็จะตอบกลับไปยังโปรเซสที่เรียกใช้ ระบบไฟล์จึงเปรียบเสมือนกับเน็ตเวิร์คไฟล์เซิฟเวอร์ (Network File Server)

การจัดการกับข่าวสารในระบบไฟล์

ระบบไฟล์บริการชนิดข่าวสารทั้งหมด 29 ชนิด โดยมีชนิดของข่าวสาร,พารามิเตอร์และผลลัพธ์ ดังตาราง โปรเซสจะทำการวนลูปคอยข่าวสารจากโปรเซสอื่นและเรียกใช้ฟังก์ชันที่เกี่ยวข้องตามชนิดของข่าวสารที่ได้รับ หลังจากนั้นจึงส่งสถานะกลับในรูปของข่าวสารเช่นกัน แล้วจึงกลับไปคอยรับข่าวสารต่อไป ดังตาราง 2.2

ชนิดของข่าวสาร	อินพุทพารามิเตอร์	ค่าตอบกลับ
ACCESS	ชื่อไฟล์, โหมดการติดต่อ	สถานะ
CHDIR	ชื่อของไดเรกทอรีใหม่	สถานะ
CHMOD	ชื่อไฟล์, โหมดใหม่	สถานะ
CHOWN	ชื่อไฟล์, ชื่อเจ้าของ, ชื่อกลุ่ม	สถานะ
CHROOT	ชื่อของไดเรกทอรีรากใหม่	สถานะ
CLOSE	ไฟล์เดสคริปเตอร์ที่ต้องการปิด	สถานะ
CREAT	ชื่อของไฟล์ที่จะสร้าง,โหมด	ไฟล์เดสคริปเตอร์
DUP	ไฟล์เดสคริปเตอร์ (DUP2 ใช้ 2 ตัว)	ไฟล์เดสฯใหม่
FSTAT	ชื่อไฟล์,บัฟเฟอร์ที่จะใช้เก็บสถานะ	สถานะ
IOCTL	ไฟล์เดสคริปเตอร์,ฟังก์ชัน,อาร์กิวเมนต์	สถานะ
LINK	ชื่อของไฟล์ที่จะลิงค์, ชื่อลิงค์	สถานะ
LSEEK	ไฟล์เดสคริปเตอร์,ออฟเซต,ตำแหน่งเริ่มต้น	ตำแหน่งใหม่
MKNOD	ชื่อของไดเรกทอรีหรือไฟล์พิเศษ,โหมด,แอดเดรส	สถานะ
MOUNT	ไฟล์พิเศษ, ตำแหน่งที่จะติดตั้ง, แฟลกอ่าน/เขียน	สถานะ

ตาราง 2.2 แสดงบริการที่ส่วนระบบไฟล์สนับสนุน

ชนิดของข่าวสาร	อินพุทพารามิเตอร์	ค่าตอบกลับ
OPEN	ชื่อของไฟล์ที่จะเปิด, แฟลกอ่าน/เขียน	ไฟล์เดสคริปเตอร์
PIPE	ไม่มี	ไฟล์เดสคริปเตอร์
READ	ไฟล์เดสคริปเตอร์, บัฟเฟอร์, จำนวนไบต์	จำนวนไบต์ที่อ่าน
STAT	ชื่อไฟล์, บัฟเฟอร์เก็บสถานะ	สถานะ
STIME	พอยน์เตอร์ไปยังเวลาปัจจุบัน	สถานะ
SYNC	ไม่มี	OK
TIME	พอยน์เตอร์สำหรับเก็บเวลาปัจจุบัน	เวลาปัจจุบัน
TIMES	บัฟเฟอร์สำหรับเก็บเวลาของโปรเซสและของลูก	สถานะ
UMASK	คอมพลีเมนต์ของโหมดมาสค์	OK
UMOUNT	ชื่อของไฟล์พิเศษที่จะยกเลิกการติดตั้ง	สถานะ
UNLINK	ชื่อไฟล์ที่จะลบ	สถานะ
UTIME	ชื่อไฟล์, เวลาของไฟล์	OK
WRITE	ไฟล์เดสคริปเตอร์, บัฟเฟอร์, จำนวนไบต์	จำนวนไบต์ที่เขียน
REVIVE	โปรเซสที่จะนำกลับมาใช้	ไม่ตอบกลับ
UNPAUSE	โปรเซสที่จะตรวจ	---

หมายเหตุ สถานะจะตอบกลับอยู่สองค่าคือ OK หรือ ERROR และชื่อไฟล์จะใช้พอยน์เตอร์ชี้

ตาราง 2.2 แสดงบริการที่ส่วนระบบไฟล์สนับสนุน (ต่อ)

บล็อคแคช (BLOCK CACHE)

แคชในมินิกซ์จะอยู่ในรูปของบัฟเฟอร์ของอะเรย์ ซึ่งแคชแต่ละอันจะมีเฮดเดอร์ที่เป็นโครงสร้างดังนี้

```
struct buf {
        /* ส่วนข้อมูลของบัฟเฟอร์ */
    union {
        char b_data[BLOCK_SIZE];        /* จุดเริ่มต้นดาต้าของผู้ใช้ */
        dir_struct b_dir[NR_DIR_ENTRIES]; /* บล็อกไดเร็คตอรี */
        zone_nr b_ind[NR_INDIRECTS];  /* บล็อกอินไดเร็คตอรี */
        d_inode b_inode[INODES_PER_BLOCK]; /* บล็อกไอโนด */
        int b_int[INTS_PER_BLOCK];    /* บล็อกของอินทิเจอร์ */
    } b;
    /* ส่วนเฮดเดอร์ของบัฟเฟอร์ */
    struct buf *b_next;         /* พอยน์เตอร์ชี้ไปยังบัฟเฟอร์ถัดไป */
    struct buf *b_prev;         /* พอยน์เตอร์ชี้ไปยังบัฟเฟอร์ก่อนหน้านี้ */
    struct buf *b_hash;         /* พอยน์เตอร์บนแฮชเชน */
    block_nr b_blocknr;         /* หมายเลขบล็อกของอุปกรณ์ */
    dev_nr b_dev;               /* หมายเลขอุปกรณ์*/
    char b_dirt;                /* สถานะแสดง CLEAN หรือ DIRTY */
    char b_count;               /* จำนวนผู้ใช้ของบัฟเฟอร์นี้ */
} buf[NR_BUFS];
```

บล็อคแคชทำหน้าที่ในการเก็บข้อมูลที่ได้อ่านจากดิสค์ไว้ในหน่วยความจำ เท่าที่จะสามารถจัดเก็บได้ขึ้นอยู่กับขนาดของแคช โดยแคชจะแบ่งออกเป็นบล็อคๆที่มีขนาดเท่าๆกัน โดยจะแมปไปยังบล็อคของดิสค์ที่ได้อ่านมา เมื่อโปรเซสใดต้องการอ่านดิสค์ที่มีหมายเลขของบล็อคตรงกันกับหมายเลขของบล็อคที่เก็บอยู่ในบล็อคแคช ระบบไฟล์ก็จะลอกข้อมูลจากบล็อคแคชไปยังโปรเซสที่ต้องการอ่านทันที โดยไม่ต้องติดต่อกับดิสค์จริงๆ

บทที่ 3

อธิบายฟังก์ชันต่างๆในระบบปฏิบัติการมินิกซ์

ในบทนี้จะกล่าวถึงรายละเอียดต่างๆ ของฟังก์ชันที่ได้ศึกษาในระบบปฏิบัติการมินิกซ์ สำหรับใช้ในอ้างอิงประกอบความเข้าใจในภายหลัง เพื่อนำไปใช้ในการพัฒนาต่อไป การอธิบายการทำงานของฟังก์ชันจะแบ่งเป็นกลุ่มๆ ตั้งแต่ส่วนเคอร์นัล ส่วนจัดการหน่วยความจำ และส่วนระบบไฟล์ โดยมีรูปแบบการอธิบายดังนี้

ชื่อไฟล์

ฟังก์ชัน	`function_name(var1, var2, ...)`
อินพุท	`type var1;` /* แสดงพารามิเตอร์ต่างๆของอินพุทว่ามีอะไรบ้าง */
	`type var2;` /* และแต่ละตัวมีชนิดของข้อมูลเป็นอย่างไร */
เอาท์พุท	แสดงค่าที่ผ่านกลับไปยังฟังก์ชันที่เรียกใช้ฟังก์ชันนี้
หน้าที่	อธิบายหน้าที่ของฟังก์ชันนี้
การทำงาน	อธิบายขั้นตอนการทำงานของฟังก์ชันทีละขั้นเป็นข้อๆ
ผู้เรียกใช้	กล่าวถึงฟังก์ชันที่เป็นผู้เรียกใช้ฟังก์ชันนี้

ฟังก์ชันต่าง ๆ ในการจัดสรรเวลาในระบบมินิกซ์

ไฟล์ kernel/proc.c

ฟังก์ชัน pick_proc

หน้าที่ นำ process จากหัวคิวมารัน

การทำงาน 1. ตรวจสอบคิวในแต่ละ level โดยเรียงจาก TASK_Q, SERVER_Q และ USER_Q ว่ามี process ไหนต้องการใช้ CPU บ้าง

2. เก็บ process เดิมไว้ในตัวแปร prev_proc

3. ตรวจสอบ hdy_head ว่าเป็น NIL_PROC หรือไม่

ถ้าไม่เป็น NIL_PROC แสดงว่ามี process อื่นต้องการใช้ CPU ให้คำนวณหา cur_proc และหา proc_ptr และถ้าหาก cur_proc มีค่ามากกว่า LOW_USER แล้วprocess นี้จึงสามารถใช้ CPU ได้ โดยการเซต bill_ptr ให้มีค่าเท่ากับ proc_ptr

ถ้าเป็น NIL_PROC แสดงว่าไม่มี process ใดต้องการใช้ CPU เลย ดังนั้นเราจึงได้เซตให้ cur_proc = IDLE และ proc_ptr และ bill_ptr เป็นโปรเซสแอดเดรสของ HARDWARE

ไฟล์ kernel/proc.c

ฟังก์ชัน ready

อินพุท struct proc *rp; /* โปรเซสที่จะนำไปใส่ไว้ในคิวพร้อมที่จะทำงาน */

หน้าที่ นำ proc ลงในคิว

การทำงาน 1. คำนวณหา process number

2. ตรวจสอบ process ว่าอยู่ใน level อะไร

3. นำ process ต่อลงที่ท้ายของคิวนั้น

ฟังก์ชันต่างๆในส่วนอินเตอร์รัพท์

ไฟล์ kernel/main.c

ฟังก์ชัน main()

หน้าที่ เป็นฟังก์ชันแรกสุดของส่วนเคอร์นัล (kernel) ทำหน้าที่ในการเซตค่าพารามิเตอร์เริ่มแรกของระบบ

การทำงาน
1. เซตตารางโปรเซสต่างๆทั้งหมด ไม่ว่าจะเป็นไอโอทาสค์, เซิร์ฟเวอร์, โปรเซสของผู้ใช้ และทำการใส่ทาสค์และเซิร์ฟเวอร์ (MM, FS, INIT) เข้าไปในคิวเตรียมทำงาน (ready queue) เซตแมปของหน่วยความจำแต่ละทาสค์และเซิร์ฟเวอร์
2. เซต p_flags ให้เท่ากับ P_SLOT_FREE สำหรับโปรเซสของผู้ใช้ที่ยังไม่เกิด
3. เก็บชนิดของจอภาพ
4. เซตอินเตอร์รัพท์เวกเตอร์ของชิสเต็มคอลล์, นาฬิกา, เทอร์มินัล, ดิสค์, พรินเตอร์ ไปที่แอดเดรสของฟังก์ชัน s_call, clock_int, tty_int, disk_int, lpr_int ตามลำดับ
5. เซตอินเตอร์รัพท์เวกเตอร์ที่เหลือ โดยให้อินเตอร์รัพท์หมายเลขน้อยกว่าสิบหกชี้ไปที่แอดเดรสของฟังก์ชัน surprise ในขณะอินเตอร์รัพท์หลังจากนั้นชี้ไปที่ฟังก์ชัน trp
6. เก็บแอดเดรสของตารางโปรเซส (ตัวแปร proc) ไว้ในแอดเดรสที่แน่นอน
7. เลือกโปรเซสในคิวเตรียมทำงานขึ้นมา ด้วยฟังก์ชัน pick_proc
8. เข้าสู่ตัวจัดสรรการทำงาน (scheduler) ด้วยฟังก์ชัน restart

ผู้เรียก ฟังก์ชันนี้ถูกเรียกใช้โดยฟังก์ชัน MINIX ในแอสเซมบลิไฟล์ kernel/mpx88.s

ไฟล์ kernel/proc.c

ฟังก์ชัน interrupt(int task, message *m_ptr)

อินพุท int task; /* หมายเลขทาสค์ที่จะส่งข่าวสารไปให้ (มีค่าเป็นลบเสมอ) */

```
message *m_ptr;                    /* พอยน์เตอร์ชี้ไปที่ข่าวสารที่จะส่ง */
```

หน้าที่ ส่งข่าวสารไปยังทาสค์ที่กำหนด ถ้าส่งไม่สำเร็จให้เซตบิตใน busy_map ของทาสค์นั้นเป็นหนึ่ง และเก็บตำแหน่งของข่าวสารนั้นไว้ หลังจากการส่งแล้วจะเช็คว่าทุกๆ ทาสค์ว่ามีทาสค์ใดที่ยังรอการส่งอยู่ ก็ให้ส่งข่าวสารของทาสค์นั้น เสร็จแล้วจึงเลือกโปรเซสขึ้นมาทำงานต่อไปด้วยฟังก์ชัน pick_proc

ผู้เรียก ฟังก์ชันนี้ถูกเรียกใช้โดย ISR ของดิสค์ (disk_int), นาฬิกา (clock_int), คีย์บอร์ด (keyboard)

ไฟล์ kernel/mpx88.s

ฟังก์ชัน MINIX

หน้าที่ เป็นรูทีนภาษาแอสเซมบลีแรกของส่วน kernel มีหน้าที่เซตเซกเมนต์รีจิสเตอร์ต่างๆและสแต็กแล้วจึงไปเรียกฟังก์ชัน main

---

ไฟล์ kernel/mpx88.s

ฟังก์ชัน s_call

หน้าที่ เป็น ISR ของฟังก์ชันของระบบ (system call) ทำหน้าที่ในการให้บริการสำหรับฟังก์ชันต่างๆแก่ผู้เรียกใช้ ใน MINIX จะสนับสนุนการรับส่งข่าวสาร ซึ่งมีโหมดการติดต่อข่าวสารเป็นรับ, ส่งหรือรับส่งทั้งคู่ โดยฟังก์ชันนี้จะไปเรียกใช้ sys_call ในการทำหน้าที่รับส่งข่าวสาร

ไฟล์ kernel/mpx88.s

ฟังก์ชัน tty_int

หน้าที่ เป็น ISR ของฮาร์ดแวร์เมื่อมีการเกิดอินเตอร์รัพท์จากการกดหรือปล่อยคีย์บอร์ด โดยจะไปเรียกใช้ฟังก์ชัน keyboard

ไฟล์ kernel/mpx88.s

ฟังก์ชัน lpr_int

หน้าที่ เป็นทางเข้า ISR สำหรับใช้ในการส่งข้อมูลออกทางเครื่องพิมพ์ โดยจะไปเรียกใช้ฟังก์ชัน pr_char

ไฟล์ kernel/mpx88.s

ฟังก์ชัน disk_int

หน้าที่ เป็น ISR ของฮาร์ดแวร์เมื่อมีการเกิดอินเตอร์รัพท์จากดิสค์ไดรฟ์ โดยจะทำการสร้างข่าวสารให้มีชนิดเป็น FLOPPY ฟังก์ชันย่อยเป็น DISKINT ก่อนที่จะไปเรียกใช้ฟังก์ชัน interrupt

ไฟล์ kernel/mpx88.s

ฟังก์ชัน clock_int

หน้าที่ เป็น ISR ของฮาร์ดแวร์เมื่อมีการเกิดอินเตอร์รัพท์จาก timer tick เกิดขึ้นด้วยความถี่ 60 เฮิร์ทโดยจะทำการสร้างข่าวสารให้มีชนิดเป็น CLOCK ฟังก์ชันย่อยเป็น CLOCK_TICK ก่อนที่จะไปเรียกใช้ฟังก์ชัน interrupt

ไฟล์ kernel/mpx88.s

ฟังก์ชัน surprise

หน้าที่ เป็นทางเข้าของ ISR สำหรับดักจับฮาร์ดแวร์อินเตอร์รัพท์ที่ไม่คาดหวัง ซึ่งมีหมายเลขอินเตอร์รัพท์น้อยกว่า 16 โดยจะไปเรียกใช้ฟังก์ชัน unexpected_int

ไฟล์ kernel/mpx88.s

ฟังก์ชัน trp

หน้าที่ เป็นทางเข้าของ ISR สำหรับดักจับซอฟท์แวร์อินเตอร์รัพท์ (คำสั่ง INT) ที่ไม่สนับสนุนใน MINIX โดยมีหมายเลขอินเตอร์รัพท์ตั้งแต่ 16 ขึ้นไปจนถึง 255 โดย

จะไปเรียกใช้ฟังก์ชัน trap

ไฟล์ kernel/mpx88.s

ฟังก์ชัน sys_call(function, caller, src_dest, m_ptr)

อินพุท
```
int function;     /* ชนิดฟังก์ชันมีค่าระหว่าง SEND, RECEIVE, BOTH */
int caller;       /* ผู้เรียกใช้ฟังก์ชันของซิสเต็ม */
int src_dest;     /* ต้นทางที่กำลังคอยข่าวสารหรือ */
                  /* ปลายทางที่จะส่งข่าวสารไปให้ */
message *m_ptr;   /* พอยน์เตอร์ไปที่ข่าวสาร */
```

หน้าที่ เป็นฟังก์ชันที่คอยบริการให้แก่โปรเซสในระบบ โดย MINIX สนับสนุนฟังก์ชันเฉพาะการรับส่งข่าวสารเท่านั้น สำหรับโปรเซสของผู้ใช้สามารถใช้ฟังก์ชัน BOTH ได้อย่างเดียวเท่านั้น โดยการส่งข่าวสารจะไปเรียกฟังก์ชัน mini_send ในขณะที่การรอรับข่าวสารจะไปเรียกฟังก์ชัน mini_rec

---

ฟังก์ชันต่างๆในส่วนแรมดิสค์

ไฟล์ kernel/memory.c

ฟังก์ชัน mem_task

หน้าที่ เป็น main โปรแกรมของ disk driver

การทำงาน 1. ทำการเซตค่าเริ่มต้นของ task ได้แก่

- การเซตจุดเริ่มต้นของ kernel โดยใส่ในตัวแปร ram_orgin [KMEM_DEV]
- การเซตขนาดของ kernel โดยใส่ในตัวแปร ram_limit [KMEM_DEV]
- การเซตขนาดของ absolute memory โดยใส่ในตัวแปร ram_limit

[MEM_DEV]

2. ทำ loop ของ memory task

- รอรับ message
- ตรวจสอบ message (mess.m_type) ที่รับมาว่าเป็นประเภท
  1. DISK_READ จะเรียกฟังก์ชัน do_mem
  2. DISK_WRITE จะเรียกฟังก์ชัน do_mem
  3. DISK_IOCTL จะเรียกฟังก์ชัน do_setup
  4. อื่นๆ แสดงว่ามีความผิดพลาด

3. ตอบรับ message

ไฟล์ kernel/memory.c

ฟังก์ชัน do_mem(m_ptr)

อินพุท register message *m_ptr;

หน้าที่ กำหนดตำแหน่งที่จะไปเขียนหรืออ่านข้อมูลจาก user physical มายัง memory physical หรือจาก memory physical มายัง user physical ฟังก์ชันนี้จะถูกเรียกโดย mem_task

การทำงาน 1. ตรวจสอบ device ว่าเป็น ram, kmem, mem หรือ null ถ้าไม่ใช่ทั้ง 4 อย่าง ก็จะ return ค่าความผิดพลาด แต่ถ้าเป็น null ก็จะ return 0 ถ้าเป็นการอ่านดิสค์ return COUNT ถ้าเป็นการเขียนดิสค์

2. กำหนดค่า memory physical ใส่ในตัวแปร mem_phys โดยจะมีลำดับดังนี้
   - ตรวจสอบตำแหน่งที่จะอ่าน-เขียน (ได้จาก m_ptr->POSITION)
   - นำตำแหน่งที่อ่าน-เขียน บวกกับจุดเริ่มต้นของแรมของdevice ถ้าบวกแล้วเกิน ram_limit[device] ก็ให้ return 0
   - ตรวจสอบจำนวนที่จะอ่านว่าเกิน ram_limit หรือไม่

3. กำหนดแอดเดรสของข้อมูลที่ต้องการจะอ่านหรือเขียน

4. ทำการก๊อปปี้ข้อมูล

- ถ้าเป็นการอ่านดิสค์ ก็อปปี้จาก mem_phys ไป user_phys
- ถ้าเป็นการเขียนดิสค์ ก็อปปี้จาก user_phys ไป mem_phys

ไฟล์ kernel/memory.c

ฟังก์ชัน do_setup(m_ptr)

อินพุท register message *m_ptr;

หน้าที่ ทำการกำหนดจุดเริ่มต้นและจุดสิ้นสุดของ RAM disk

การทำงาน 1. ตรวจสอบ device ว่าเป็น ram, kmem, mem หรือ null ถ้าไม่ใช่ทั้ง 4 อย่าง ก็จะ return ค่าความผิดพลาด

2. กำหนดค่าเริ่มต้นและขนาดของ RAM

ฟังก์ชันต่าง ๆ ในส่วนของซิสเต็มทาสค์

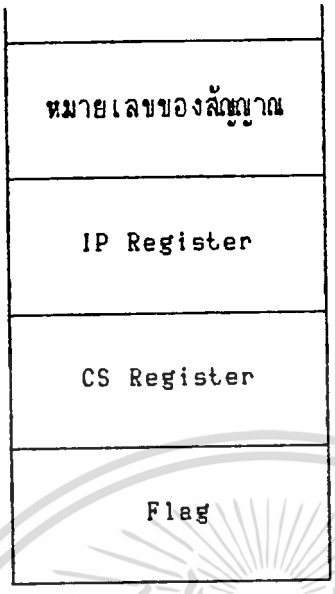
ไฟล์ kernel/system.c

ฟังก์ชัน do_sig(m_ptr)

อินพุท message *m_ptr;

เอาท์พุท สถานะของการทำงานว่า OK หรือไม่

หน้าที่ ทำการผ่านค่าต่าง ๆ ลงในสแต็กของโปรเซสที่จะรับสัญญาณดังรูป 3.1 และเปลี่ยนค่าของฟิลด์ p_sp เป็นค่าของสแต็กพอยน์เตอร์หลังจากผ่านค่าลงสแต็กแล้ว และ p_pcpsw.pc เป็นแอดเดรสของรูทีนที่จะจัดการสัญญาณ เพื่อว่าเมื่อโปรเซสนี้ทำงานอีกครั้งหนึ่ง ก็จะกระโดดไปทำที่รูทีนนั้น ๆ ทันที



รูป 3.1 โครงสร้างของสแต็กของโปรเซสที่จะรับสัญญาณ

การทำงาน

1. ทำการก๊อปปี้ค่าหมายเลขสัญญาณ, IP, CS และ Flag ไปใส่ตัวแปรที่ชื่อว่า sig_stuff ซึ่งมีขนาด 8 ไบต์พอดี โดยการเรียกฟังก์ชัน build_sig ซึ่งเป็นภาษาแอสเซ็มบลี อยู่ในไฟล์ kernel/klib88.s
2. ทำการก๊อปปี้ค่าจาก sig_stuff ไปที่สแต็กของโปรเซสที่จะรับสัญญาณ
3. เซ็ตค่าของฟิลด์ p_sp และ p_pcpsw.pc ตามที่กล่าวมาแล้ว

ผู้เรียกใช้ ฟังก์ชัน sys_task ในไฟล์ kernel/system.c

ไฟล์ kernel/system.c

ฟังก์ชัน do_copy(m_ptr)

อินพุท message *m_ptr;

เอาท์พุท สถานะของการทำงานว่า OK หรือไม่

หน้าที่ ทำการก๊อปปี้ค่า ระหว่างยูสเซอร์โปรเซส กับระบบไฟล์ หรือระหว่างยูสเซอร์โปรเซส กับส่วนจัดการหน่วยความจำ

การทำงาน

1. ถ้าหมายเลขโปรเซส ไม่เท่ากับ ABS ทำการแปลงจากแอดเดรสเสมือนเป็น

ฟิสิคัลแอดเดรส ทั้งโปรเซสต้นทาง และโปรเซสปลายทาง โดยการเรียกใช้ฟังก์ชัน umap ในไฟล์ kernel/system.c

2. ทำการก๊อปปี้ข้อมูล

ผู้เรียกใช้ ฟังก์ชัน sys_task ในไฟล์ kernel/system.c

ไฟล์ kernel/system.c

ฟังก์ชัน `cause_sig(proc_nr, sig_nr)`

อินพุท
```
int proc_nr;        /* หมายเลขโปรเซสที่จะส่งสัญญาณไปให้ */
int sig_nr;         /* หมายเลขของสัญญาณ */
```

เอาท์พุท ไม่มี

หน้าที่ เซ็ตบิตในฟิลด์ p_pending ในสล้อตของโปรเซสเทเบิลของโปรเซสที่ต้องการส่งสัญญาณไปให้ โดยจะเป็นไปตามหมายเลขสัญญาณ กล่าวคือ ถ้าสัญญาณหมายเลข 0 ก็จะเซ็ตบิต 0 ให้เป็น 1 และสัญญาณหมายเลข 1 ก็จะเซ็ตบิต 2 เป็นเช่นนี้เรื่อยไป ทั้งหมด 16 สัญญาณ และเมื่อเซ็ตแล้ว ก็ทำการบอกส่วนจัดการหน่วยความจำให้จัดการสัญญาณ โดยการเรียกใช้ฟังก์ชัน inform ซึ่งอยู่ในไฟล์ kernel/system.c

ผู้เรียกใช้ ทีทีวายทาสก์ ส่งสัญญาณ SIGINT เมื่อได้รับ DEL (ฟังก์ชัน in_char ในไฟล์ kernel/tty.c)

คล็อกทาสก์ ส่งสัญญาณ SIGALRM เมื่อครบเวลาที่ได้ตั้งไว้ (ฟังก์ชัน do_clocktick ในไฟล์ kernel/clock.c)

ไฟล์ kernel/system.c

ฟังก์ชัน `inform(proc_nr)`

อินพุท
```
int proc_nr;        /* MM_PROC_NR หรือ FS_PROC_NR เท่านั้น */
```

เอาท์พุท ไม่มี

หน้าที่ ตรวจดูว่าส่วนจัดการหน่วยความจำ ว่างหรือไม่ ถ้าว่าง ก็จะทำการส่งข่าวสารให้

ทำการจัดการเกี่ยวกับการส่งสัญญาณต่าง ๆ

การทำงาน

1. ตรวจดูว่าส่วนจัดการหน่วยความจำ ว่างหรือไม่ ถ้าไม่ว่างก็สิ้นสุดการทำงาน
2. วนลูป ตรวจทุกโปรเซสในหน่วยความจำ ยกเว้นโปรเซสของเคอร์เนล ว่ามีโปรเซสใด ได้รับสัญญาณบ้าง โดยจะดูจากโปรเซสเทเบิล ถ้ามีก็จะทำการส่งข่าวสารให้ส่วนจัดการหน่วยความจำ เพื่อให้ทำการจัดการกับสัญญาณนั้น ๆ แล้วสิ้นสุดการทำงานของฟังก์ชัน

ไฟล์ kernel/system.c

ฟังก์ชัน `umap(rp, seg, vir_addr, bytes)`

อินพุท
```
struct proc *rp;      /* แอดเดรสของสล้อตของโปรเซส */
int seg;              /* ชนิดของเซ็กเมนต์ T, D หรือ S */
vir_bytes vir_addr;   /* แอดเดรสเสมือน */
vir_bytes bytes;      /* ขนาดของหน่วยความจำที่จะอ้างถึง */
```

เอาท์พุท พิสิคัลแอดเดรส หรือถ้าไม่สามารถแปลงได้ จะรีเทิร์นค่า 0

หน้าที่ ทำการแปลงแอดเดรสเสมือน ไปเป็นพิสิคัลแอดเดรส

การทำงาน

1. ตรวจดูว่าโปรเซสนั้น ๆ สามารถอ้างหน่วยความจำที่ระบุ ได้หรือไม่ โดยจะพิจารณาขนาดของหน่วยความจำที่จะอ้างถึงด้วย ถ้าไม่สามารถอ้างได้ ให้รีเทิร์นค่า 0
2. ทำการแปลงจากแอดเดรสเสมือน ไปเป็นพิสิคัลแอดเดรส

ผู้เรียกใช้ ยูทิลิตี้ทั่วไป

ฟังก์ชันการทำงานต่าง ๆ ในส่วนการจัดการหน่วยความจำ

ไฟล์ mm/main.c

ฟังก์ชัน main ()

อินพุท ไม่มี

เอาท์พุท ไม่มี

หน้าที่ เป็นเมนโปรแกรมหลักของการจัดการหน่วยความจำ

การทำงาน

1. เซตค่าเริ่มต้นของตารางหน่วยความจำ (memory manager table)
2. เข้าสู่ลูปอนันต์ตั้งแต่บรรทัดที่ 3-10
3. จะรอข่าวสารว่ามีใครต้องการใช้หน่วยความจำบ้าง โดยการเรียกฟังก์ชัน get_work
4. หาแอดเดรสของโปรเซสที่ต้องการใช้
5. เซตค่า error เป็น OK เพื่อลบค่าความผิดพลาดของค่าเดิม
6. เซตค่า dont_reply เป็น FALSE เพื่อแสดงว่าได้มีการตอบรับแล้ว
7. ตรวจสอบผู้เรียกใช้ว่าเป็น system call หรือไม่ ถ้าไม่ใช่ เซตค่า error เท่ากับ E_BAD_CALL ถ้าใช่ ก็จะไปเรียกฟังก์ชันที่ผู้เรียกใช้ต้องการ ซึ่งภายในฟังก์ชันนั้นอาจมีการเซตค่า dont_reply หรือ error ได้ ถ้าหากมีข้อผิดพลาดเกิดขึ้น
8. ตรวจสอบค่า dont_reply ว่าใช่ค่า FALSE หรือไม่ ถ้าไม่ใช่ ให้กลับไปทำข้อ 3 ใหม่
9. ถ้าผู้เรียกใช้เท่ากับ EXEC และไม่เกิดความผิดพลาด ก็ให้กลับไปทำข้อ 3 ใหม่
10. ตอบรับกลับไปยังผู้เรียกใช้ โดยการเรียกใช้ฟังก์ชัน reply

ผู้เรียกใช้ kernel

ไฟล์ mm/main.c

ฟังก์ชัน get_work ()

อินพุท ไม่มี

เอาท์พุท mm_call /* ตัวแปรโกลบอล */

หน้าที่ รอยคอยรับข่าวสารเพื่อใช้ในการจัดการหน่วยความจำ

การทำงาน

1. รอรับข่าวสาร โดยเรียกฟังก์ชัน receive
2. ตรวจสอบดูว่าใครส่งข่าวสารมา ถ้าหากเป็นทาสค์หรือมีจำนวนมากกว่า NR_PROS ก็ให้ออกจากระบบโดยการเรียกฟังก์ชัน panic
3. เซตค่า mm_call เป็นผู้เรียกใช้จาก mm_in.m_type

ผู้เรียกใช้ main

ไฟล์ mm/main.c

ฟังก์ชัน reply (proc_nr, result, res2, respt)

อินพุท
```
int proc_nr;    /* โปรเซสที่จะตอบกลับ */
int result;     /* ผลลัพธ์ของการเรียกใช้ (ปกติเป็นค่า OK หรือค่าความ
                   ผิดพลาด) */
int res2;       /* ผลลัพธ์รอง */
char *respt;    /* พอยน์เตอร์ของผลลัพธ์ */
```

เอาท์พุท ไม่มี

หน้าที่ ส่งผลลัพธ์กลับไปยังโปรเซสของผู้ใช้

การทำงาน

1. หาแอดเดรสของโปรเซสที่จะตอบกลับ
2. ตรวจสอบค่าสถานะของหน่วยความจำ ถ้าเป็น IN_USE หรือ HANGING แล้วกลับไปยังโปรแกรมที่เรียกใช้
3. เซตค่าผลลัพธ์ต่างๆ ลงในข่าวสารที่จะตอบรับกลับไป

4. ทำการตอบกลับไปยังผู้เรียกใช้ โดยเรียกฟังก์ชัน send

ผู้เรียกใช้ main

ไฟล์ mm/main.c

ฟังก์ชัน mm_init ()

อินพุท ไม่มี

เอาท์พุท ไม่มี

หน้าที่ ส่งผลลัพธ์กลับไปยังโปรเซสของผู้ใช้

การทำงาน

1. หาหน่วยความจำทั้งหมดจากฟังก์ชัน get_tot_mem
2. เซตค่าเริ่มต้นในตารางหน่วยความจำ โดยเรียกฟังก์ชัน mem_init
3. เซตค่าในตารางหน่วยความจำให้เป็น IN_USE
4. เซตค่า procs_in_use เป็น 3 เนื่องจากมีโปรเซส 3 โปรเซส คือ MM, FS และ INIT
5. กำหนดขนาดของสแต็ก

ผู้เรียกใช้ main

ไฟล์ mm/main.c

ฟังก์ชัน do_brk2 ()

อินพุท ไม่มี

เอาท์พุท ไม่มี

หน้าที่ จะถูกเรียกโดย FS ในระหว่างการเซตค่าเริ่มต้น ฟังก์ชันนี้จะถูกเรียกเพียงครั้งเดียวเท่านั้น ฟังก์ชันนี้จะใช้ในการหาจุดเริ่มต้นและขนาดของหน่วยความจำ เช่น ขนาดของโปรเซส INIT, จำนวนหน่วยความจำ เป็นต้น

การทำงาน

1. ตรวจดูว่า ถ้าไม่ใช่ FS เป็นผู้เรียกใช้ก็ให้ออกจากฟังก์ชันนี้

2. กำหนดและหาขนาดของเท็กซ์, ดาต้า, ขนาดของหน่วยความจำ, จุดเริ่มต้นของแรม, ขนาดของแรม
3. ทำการจองหน่วยความจำทั้งหมด โดยเรียกใช้ฟังก์ชัน alloc_mem
4. ทำการพิมพ์ผลลัพธ์ของหน่วยความจำออกมาใช้เห็น
5. เซตค่าเริ่มต้นตารางของ INIT

ผู้เรียกใช้ FS

ไฟล์ mm/main.c

ฟังก์ชัน `set_map (proc_nr, base, clicks)`

อินพุท
```
int proc_nr;            /* หมายเลขของโปรเซส */
phys_clicks  base;      /* จุดเริ่มต้นของโปรเซส */
phys_clicks  clicks;    /* ขนาดที่จะเซต หน่วยเป็น คลิก */
```

เอาท์พุท ไม่มี

หน้าที่ จะทำการเซตค่าจุดเริ่มต้นในหน่วยความจำของโปรเซสนั้น

การทำงาน

1. เซตค่าการแมปหน่วยความจำต่างๆลงในตารางของโปรเซสนั้น
2. จัดการให้โปรเซสชี้ไปยังตำแหน่งเซกเมนต์ของมันเอง โดยเรียกใช้ฟังก์ชัน sys_newmap

---

ไฟล์ mm/signal.c

ฟังก์ชัน `do_signal()`

อินพุท ไม่มี

เอาท์พุท ไม่มี

หน้าที่ เช็ตว่าต้องการจะรับ หรือไม่สนใจ สัญญาณใด ๆ หรือเช็ตว่ารูทีนใด จะจัดการเมื่อได้รับสัญญาณ

การทำงาน

1. ตรวจดูว่าสัญญาณที่ต้องการเซ็ตเป็น SIGKILL หรือไม่ ถ้าใช่ ก็ให้สิ้นสุดการทำงานของฟังก์ชัน เพราะว่าสัญญาณ SIGKILL ยูสเซอร์โปรเซสไม่สามารถเซ็ตเป็นไม่สนใจ และไม่อนุญาติให้จัดการเองด้วย
2. ถ้าฟังก์ชันที่ส่งมาในข่าวสารเป็น SIG_IGN ก็จะทำการเซ็ตบิตในฟิลด์ mp_ignore และเคลียบิตในฟิลด์ mp_catch ในโปรเซสเทเบิลสล้อตของส่วนจัดการหน่วยความจำ โดยการเลือกบิต จะใช้วิธีเดียวกับที่ได้อธิบายไปในฟังก์ชัน cause_sig หลังจากนั้นก็จะสิ้นสุดการทำงานของฟังก์ชัน
3. ถ้าฟังก์ชันที่ส่งมาในข่าวสารเป็น SIG_DFL ก็จะทำการเคลียบิตในฟิลด์ mp_ignore และเคลียบิตในฟิลด์ mp_catch ในโปรเซสเทเบิลสล้อตของส่วนจัดการหน่วยความจำ โดยการเลือกบิต จะใช้วิธีเดียวกับที่ได้อธิบายไปในฟังก์ชัน cause_sig หลังจากนั้นก็จะสิ้นสุดการทำงานของฟังก์ชัน
4. ถ้าไม่ใช่กรณีที่กล่าวมา ก็จะทำการเคลียบิตในฟิลด์ mp_ignore และเซ็ตบิตในฟิลด์ mp_catch ในโปรเซสเทเบิลสล้อตของส่วนจัดการหน่วยความจำ โดยการเลือกบิต จะใช้วิธีเดียวกับที่ได้อธิบายไปในฟังก์ชัน cause_sig หลังจากนั้นก็จะสิ้นสุดการทำงานของฟังก์ชัน

ผู้เรียกใช้ mm/main.c โดยเรียกผ่าน (*mm_callvec[])()

ไฟล์ mm/signal.c

ฟังก์ชัน do_kill()

อินพุท ไม่มี

เอาท์พุท ไม่มี

หน้าที่ ทำการทำลายโปรเซส โดยการเรียก check_sig ซึ่งอยู่ในไฟล์ mm/signal.c

ผู้เรียกใช้ mm/main.c โดยเรียกผ่าน (*mm_callvec[])()

ไฟล์ mm/signal.c

ฟังก์ชัน do_ksig()

อินพุท ไม่มี

เอาท์พุท สถานะการทำงานของฟังก์ชัน ว่า OK หรือไม่

หน้าที่ ทำการจัดการสัญญาณที่ส่งมาจากส่วนเคอร์เนล หรือส่วนระบบไฟล์ (ฟังก์ชัน inform ซึ่งอยู่ในไฟล์ kernel/system.c) โดยจะเรียกฟังก์ชัน check_sig ในไฟล์ mm/signal.c อีกที

การทำงาน

1. ตรวจดูว่าผู้ส่งสัญญาณเป็นส่วนจัดการหน่วยความจำ หรือส่วนระบบไฟล์ หรือไม่ ถ้าไม่ใช่ รีเทิร์นค่า EPERM
2. ตรวจดูว่าเป็นสัญญาณที่เกิดจากความผิดพลาดของสแต็ก (stack fault) หรือไม่ ถ้าใช่ให้ทำการเรียกฟังก์ชัน stack_fault ซึ่งอยู่ในไฟล์ mm/signal.c เพื่อทำการขยายสแต็ก แต่ถ้าไม่สามารถขยายได้ ก็จะทำการทำลายโปรเซสนั้นเสีย
3. วนลูปตรวจดูบิตแมปของสัญญาณ ดูแต่ละสัญญาณว่ามีสัญญาณใด ถูกส่งมาบ้าง ถ้ามี ก็จะทำการเรียกใช้ฟังก์ชัน check_sig โดยจะส่งหมายเลขโปรเซสไปให้ โดยกรณีที่เป็นสัญญาณ SIGINT และ SIGQUIT จะใช้หมายเลข 0 เพื่อว่าสามารถส่งสัญญาณได้พร้อม ๆ กันหลาย ๆ โปรเซส และจะส่งหมายเลขยูสเซอร์เป็น SUPER_USER
4. เซ็ตแฟล็ก dont_reply เพื่อบอก main ว่าไม่ต้องส่งข่าวสารตอบ

ผู้เรียกใช้ mm/main.c โดยเรียกผ่าน (*mm_callvec[])()

ไฟล์ mm/signal.c

ฟังก์ชัน `check_sig(proc_id, sig_nr, send_uid)`

อินพุท
```
int proc_id;    /* หมายเลขโปรเซสที่จะรับสัญญาณ หรือ 0 หรือ -1 */
int sig_nr;     /* หมายเลขสัญญาณ */
uid send_uid;   /* หมายเลขยูสเซอร์ของโปรเซสที่ทำการส่งสัญญาณ */
```

เอาท์พุท สถานะของการทำงานว่า OK หรือไม่

หน้าที่ ทำการตรวจดูว่าเงื่อนไขต่าง ๆ ของการส่งสัญญาณว่าถูกต้องหรือไม่ ถ้าไม่ก็จะรีเทิร์นค่าสถานะแสดงความผิดพลาดต่าง ๆ แต่ถ้าถูกต้อง ก็จะทำการเรียกฟังก์ชัน sig_proc เพื่อทำการส่งสัญญาณ

การทำงาน

1. วนลูปตรวจทุกยูสเซอร์โปรเซส ที่กำลังทำงานอยู่ในหน่วยความจำ โดยเงื่อนไขที่จะทำการส่งสัญญาณได้ คือ
   - ถ้าซุปเปอร์ยูสเซอร์เป็นผู้ส่งสัญญาณและหมายเลขโปรเซสเป็น -1 อนุญาติให้ทำการส่งสัญญาณได้ โดยไม่ต้องพิจารณาเงื่อนไขอื่น ๆ
   - หมายเลขยูสเซอร์ของโปรเซสที่ส่ง และรับสัญญาณ ต้องเป็นหมายเลขเดียวกัน โดยถ้าไม่ใช่ ก็จะต้องเป็นซุปเปอร์ยูสเซอร์ (superuser) เป็นผู้ส่งสัญญาณเท่านั้น นอกเหนือจากนี้ ไม่ส่ง
   - ถ้ามีการระบุหมายเลขโปรเซสที่ต้องการให้รับสัญญาณ (คือหมายเลขโปรเซสมีค่ามากกว่า 0) โปรเซสที่ระบุเท่านั้น ที่สามารถรับสัญญาณได้
   - ถ้าหมายเลขโปรเซสที่ระบุมีค่าเป็น 0 หมายความว่า ต้องการส่งสัญญาณให้โปรเซสทั้งกรุ๊ป ดังนั้น ต้องมีการตรวจหมายเลขกรุ๊ปให้ตรงกันด้วย
   - โปรเซสที่ถูกทำลายแล้ว จะไม่สามารถรับสัญญาณใด ๆ ได้
   - ถ้าสัญญาณที่จะส่งเป็นสัญญาณ SIGALRM จะเคลียบิต ALARM_ON ทิ้ง
   - ถ้าการส่งสัญญาณไม่ถูกตามเงื่อนไข หรือโปรเซสที่จะได้รับสัญญาณ ไม่ได้รับอนุญาติให้รับสัญญาณได้ ก็ให้วนลูปต่อไป
   - ถ้าโปรเซสที่จะได้รับสัญญาณกำลังทำงานเกี่ยวกับ PAUSE, WAIT, READ หรือ WRITE ให้เรียกฟังก์ชัน unpause ซึ่งอยู่ในไฟล์ mm/signal.c เพื่อที่จะรีเทิร์นค่า EINTR กลับไปที่โปรเซสนั้น
   - ทำการส่งสัญญาณ โดยเรียกฟังก์ชัน sig_proc ซึ่งอยู่ในไฟล์ mm/signal.c และถ้าโปรเซสที่ต้องการให้รับสัญญาณมีเพียงโปรเซสเดียว ก็จะหลุดออกจากลูป ถ้าไม่ใช่ ก็ทำงานในลูปต่อ
2. ถ้าโปรเซสที่ได้รับสัญญาณ ถูกทำลายไปแล้ว หรือกำลังทำงานเกี่ยวกับ PAUSE

, WAIT, READ หรือ WRITE อยู่ ก็เซ็ตแฟล็ก dont_reply

3. ถ้าทำการส่งสัญญาณได้ รีเทิรน์ค่า OK แต่ถ้าไม่ได้ รีเทิรน์ค่า ESRCH

ผู้เรียกใช้ ฟังก์ชัน do_kill และฟังก์ชัน do_ksig

ไฟล์ mm/signal.c

ฟังก์ชัน sig_proc(rmp, sig_nr)

อินพุท struct mproc *rmp;

/* พอยน์เตอร์ของสล้อตของโปรเซสเทเบิลของโปรเซสที่จะส่งสัญญาณไปให้ */

int sig_nr; /* หมายเลขของสัญญาณ */

เอาท์พุท ไม่มี

หน้าที่ ทำการส่งสัญญาณ และถ้าไม่สำเร็จ จะทำการเขียนคอร์ดัมพ์ (core dump) ลงไปบนแผ่นดิสก์ และทำลายโปรเซสปลายทางทิ้ง

การทำงาน

1. ถ้าโปรเซสปลายทาง ถูกทำลายไปแล้ว สิ้นสุดการทำงานของฟังก์ชัน
2. ถ้าโปรเซสปลายทาง ต้องการรับสัญญาณ ให้ทำการตรวจว่าสแต็กมีเนื้อที่พอที่จะทำการส่งสัญญาณหรือไม่ ถ้าพอ ก็ให้ทำการส่ง และสิ้นสุดการทำงานของฟังก์ชัน
3. ถ้าไม่สามารถส่งสัญญาณได้ ให้ทำการเขียนคอร์ดัมพ์ลงแผ่นดิสก์ และทำลายโปรเซสทิ้ง

ผู้เรียกใช้ ฟังก์ชัน check_sig

ไฟล์ mm/signal.c

ฟังก์ชัน **do_alarm()**

อินพุท ไม่มี

เอาท์พุท จำนวนวินาทีที่เหลือ

หน้าที่ ทำการตั้งเวลาปลุก (ALARM) เป็นวินาที โดยการเรียกใช้ฟังก์ชัน set_alarm ในไฟล์ mm/signal.c

ผู้เรียกใช้ mm/main.c โดยเรียกผ่าน (*mm_callvec[])()

ไฟล์ mm/signal.c

ฟังก์ชัน set_alarm(proc_nr, sec)

อินพุท int proc_nr; /* หมายเลขโปรเซสที่ต้องการตั้งเวลาปลุก */

unsigned sec; /* จำนวนวินาที */

เอาท์พุท จำนวนวินาทีที่เหลือ

หน้าที่ ส่งข่าวสารให้คล็อกทาสก์ โดยเป็นข่าวสารประเภท SET_ALARM และเซ็ตบิตในฟิลด์ mp_flags ของโปรเซสเทเบิลเป็น ALARM_ON หรือเคลียบิตถ้าเวลาที่ตั้งเป็น 0

ผู้เรียกใช้ ฟังก์ชัน do_alarm ในไฟล์ mm/signal.c

ไฟล์ mm/signal.c

ฟังก์ชัน do_pause()

อินพุท ไม่มี

เอาท์พุท ไม่มี

หน้าที่ ทำการเซ็ตบิตในฟิลด์ mp_flags ของโปรเซสเทเบิลเป็น PAUSED และเซ็ตแฟล็ก dont_reply เพื่อจะไม่ต้องส่งข่าวสารตอบ

ผู้เรียกใช้ mm/main.c โดยเรียกผ่าน (*mm_callvec[])()

ไฟล์ mm/signal.c

ฟังก์ชัน unpause(pro)

อินพุท int pro; /* หมายเลขโปรเซส */

เอาท์พุท ไม่มี

หน้าที่ ตรวจดูว่าโปรเซสที่รับสัญญาณกำลังทำงานเกี่ยวกับ WAIT, PAUSED, READ หรือ WRITE หรือไม่ ถ้าใช่ ให้ส่งข่าวสารตอบไป เพื่อให้ซิสเต็มคอลนั้น ๆ รีเทิร์นค่า

EINTR

การทำงาน 1. ตรวจดูว่าโปรเซส PAUSED อยู่หรือไม่ ถ้าใช่ก็ให้ทำการเคลียฟิลด์ mp_flags แล้วส่งข่าวสาร ให้รีเทิร์นค่า EINTR

2. ตรวจดูว่าโปรเซส WAIT อยู่หรือไม่ ถ้าใช่ก็ให้ทำการเคลียฟิลด์ mp_flags แล้วส่งข่าวสาร ให้รีเทิร์นค่า EINTR

3. ตรวจดูว่าโปรเซสกำลังทำ READ หรือ WRITE อยู่หรือไม่ โดยการเรียกใช้ฟังก์ชัน tell_fs()

ผู้เรียกใช้ ฟังก์ชัน check_sig ในไฟล์ mm/signal.c

ไฟล์ mm/signal.c

ฟังก์ชัน dump_core(rmp)

อินพุท struct mproc *rmp; /* โปรเซสที่ต้องการทำคอร์ดัมพ์ */

เอาท์พุท ไม่มี

หน้าที่ เขียนข้อมูลเกี่ยวกับโปรเซสลงในดิสก์ ได้แก่ สถานะของโปรเซส ขณะที่เกิดความผิดพลาดในการจัดการสัญญาณ, ข้อมูลในหน่วยความจำของโปรเซส ฯลฯ

ผู้เรียกใช้ ฟังก์ชัน sig_proc() ในไฟล์ mm/signal.c

ไฟล์ mm/forkexit.c

ฟังก์ชั่น do_fork()

เอาท์พุท สร้างโปรเซสลูก (child process) 1 โปรเซส

หน้าที่ จัดทำ fork system call เมื่อ user โปรเซสเรียกใช้งาน

การทำงาน 1. ตรวจสอบว่ามี memory slot ว่างหรือไม่
ถ้าไม่มี slot ที่ว่าง ให้ส่งรหัส error กลับไป

2. คำนวณจำนวนหน่วยความจำที่จะจองให้โปรเซสลูก ซึ่งเท่ากับขนาดของโปรเซสพ่อ (parent process)

3. เรียกใช้ฟังก์ชั่น alloc_mem เพื่อจองหน่วยความจำ

4. ตรวจสอบว่า มีหน่วยความจำเพียงพอให้จองหรือไม่ โดยพิจารณาจาก
   การส่งค่ากลับมาของฟังก์ชั่น alloc_mem
   ถ้าหน่วยความจำไม่เพียงพอ ให้ส่งรหัส error กลับไป
5. เรียกใช้ฟังก์ชั่น mem_copy เพื่อคัดลอกข้อมูลส่วน text,data,stack
   ของโปรเซสพ่อ ไปยัง โปรเซสลูก
6. ค้นหา memory slot ที่ว่างให้โปรเซสลูก
7. คัดลอกข้อมูลส่วน memory slot ของโปรเซสพ่อ ไปยัง โปรเซสลูก
8. จัดการเซ็ตค่าพารามิเตอร์ต่างๆใน memory slot ของโปรเซสลูก
9. คำนวณหา pid ของโปรเซสลูก
10. แจ้งให้ kernel และ file system ทราบว่า มีโปรเซสเกิดขึ้นใหม่
11. แจ้งให้โปรเซสลูกทำการ map หน่วยความจำ
12. ส่งรหัส pid ของโปรเซสลูก กลับไปยังส่วนโปรแกรมหลัก

ผู้เรียกใช้ โปรแกรมหลัก (main) ของส่วน memory management

ไฟล์ mm/forkexit.c

ฟังก์ชั่น mm_exit(rmp,exit_status)

อินพุท
```
register struct mproc *rmp ;
/* พอยต์เตอร์ชี้โปรเซสที่ทำงานเสร็จ */
int exit_status ;
/* หมายเลขแสดงสถานะของโปรเซสเมื่อทำงานเสร็จ */
```

เอาท์พุท เคลียร์โปรเซสซึ่งทำงานเสร็จแล้วออกจากระบบ

หน้าที่ จัดทำ exit system call เมื่อ user โปรเซสเรียกใช้งาน

การทำงาน
1. นำค่า status เก็บใน memory slot ของโปรเซสที่ทำงานเสร็จ
2. ตรวจสอบว่า โปรเซสพ่อ อยู่ในสถานะ waiting หรือไม่
3. ถ้าโปรเซสพ่อ อยู่ในสถานะ waiting ให้เรียกใช้ฟังก์ชั่น cleanup
4. ถ้าโปรเซสพ่อ ไม่อยู่ในสถานะ waiting ให้เซ็ตสถานะของโปรเซสลูก

เป็น hanging

5. แจ้ง kernel ให้ทราบว่าโปรเซสไม่อยู่ในระบบแล้ว

ผู้เรียกใช้ โปรแกรมหลัก (main) ของส่วน memory management

ไฟล์ mm/forkexit.c

ฟังก์ชั่น cleanup(child)

อินพุท register struct mproc *child ;

/* พอยต์เตอร์ชี้ตำแหน่งของโปรเซสลูก ที่จะลบออกจากระบบ */

เอาท์พุท โปรเซสลูก ถูกลบออกจากระบบ

หน้าที่ เคลียร์ข้อมูลต่างๆของโปรเซสลูกนั้นๆ ออกจากระบบ

การทำาน 1. คำนวณหาหมายเลข memory slot ของโปรเซสพ่อ และ โปรเซสลูก

2. คำนวณขนาดหน่วยความจำของโปรเซสลูก ที่จะลบทิ้ง

3. เรียกใช้ฟังก์ชั่น free_mem เพื่อ free หน่วยความจำ

4. เคลียร์แฟลก hanging และ in_use ของโปรเซสลูก

5. เปลี่ยนสถานะของโปรเซสพ่อ จาก waiting เป็น runing

6. ถ้าโปรเซสลูกยังมีโปรเซสย่อยอีก (โปรเซสหลาน) ให้นำโปรเซสหลาน ไปเป็นโปรเซสลูกของโปรเซส init

ผู้เรียกใช้ ฟังก์ชั่น mm_exit, ฟังก์ชั่น do_wait

ไฟล์ mm/forkexit.c

ฟังก์ชั่น do_wait()

เอาท์พุท โปรเซสลูกที่อยู่ในสถานะ hanging จะถูกลบออกจากระบบ

หน้าที่ จัดทำ wait system call เมื่อ user โปรเซสเรียกใช้งาน

การทำงาน 1. ค้นหาว่า โปรเซสพ่อ (who) มีโปรเซสลูกหรือไม่

2. ถ้าไม่มีโปรเซสลูก ให้ส่งรหัสความผิดพลาดกลับไป

2. กรณีพบโปรเซสลูก

ให้ตรวจสอบว่า โปรเซสลูกนั้นๆอยู่ในสถานะ hanging หรือไม่
ถ้าโปรเซสลูกอยู่ในสถานะ hanging ให้ลบโปรเซสลูกนั้นๆออกจาก
ระบบ โดยเรียกใช้ฟังก์ชั่น cleanup
ถ้าไม่มีโปรเซสลูก อยู่ในสถานะ hanging ให้เซ็ตสถานะของโปรเซส
พ่อเป็น waiting

ผู้เรียกใช้ โปรแกรมหลัก (main) ของส่วน memory management

ไฟล์ mm/alloc.c

ฟังก์ชัน alloc_mem (clicks)

อินพุท phys_clicks clicks; /* จำนวนคลิกที่ต้องการจะจอง */

เอาท์พุท หมายเลขคลิกที่จองได้ ถ้าจองไม่ได้จะส่ง NO_MEM กลับ

หน้าที่ จองบล็อคของหน่วยความจำจากฟรีลิสต์โดยใช้การค้นแบบเฟิร์ทฟิต (first fit)
ขนาดคลิกมีค่าเท่ากับ 16 ไบต์

การทำงาน

1. วนลูปค้นตั้งแต่ hole head จนกระทั่งสิ้นสุดลิสต์
2. ถ้าโฮลนั้นมีขนาดใหญ่กว่าที่ต้องการจะจอง ให้ส่งหมายเลขคลิกกลับ พร้อมกับลดขนาดและเปลี่ยนหมายเลขคลิกของโฮล
3. ถ้าโฮลที่ถูกจองมีขนาดเท่ากันกับที่ต้องการพอดี ให้นำโฮลนี้ไปใส่ไว้ในฟรีลิสต์
4. ถ้าไม่มีโฮลที่ต้องการ ให้ส่งค่า NO_MEM กลับ

ผู้เรียกใช้ fork, exec

ไฟล์ mm/alloc.c

ฟังก์ชัน free_mem (base, clicks)

อินพุท phys_clicks base; /* หมายเลขคลิกเริ่มต้น */
phys_clicks clicks; /* ขนาดที่ต้องการคืนหน่วยความจำ */

เอาท์พุท ไม่มี

หน้าที่ คืนหน่วยความจำเข้าสู่โฮลลิสต์ ถ้าโฮลที่ฟรีมีแอดเดรสติดกันกับโฮลข้างเคียง จะนำมารวมกันเป็นโฮลเดียวเพื่อประหยัดสล็อตของโฮล

การทำงาน

1. ถ้าไม่มีฟรีสล็อตเหลืออยู่ให้ฟ้องความผิดพลาดและออกจากระบบ
2. นำหน่วยความจำที่จะคืนนี้ไปเก็บไว้ในฟรีลิสต์ ซึ่งก็คือโฮลที่ยังไม่ได้นำมาใช้
3. เก็บโฮลนี้ไว้ในโฮลลิสต์ โดยเรียงลำดับตามหมายเลขคลิกเริ่มต้น
4. เรียกใช้ฟังก์ชัน merge เพื่อรวมโฮลที่อยู่ข้างเคียงถ้าเป็นไปได้

ผู้เรียกใช้ cleanup, new_mem

ไฟล์ mm/alloc.c

ฟังก์ชัน `del_slot (prev_ptr, hp)`

อินพุท
```
struct hole *prev_ptr;   /* พอยน์เตอร์ไปที่โฮลลิสต์ตัวก่อน */
struct hold *hp;         /* พอยน์เตอร์ชี้ไปที่โฮลที่ต้องการจะลบออก */
```

เอาท์พุท ไม่มี

หน้าที่ ลบโฮลออกจากโฮลลิสต์ โดยนำโฮลที่ลบไปใส่ไว้ในฟรีลิสต์

การทำงาน

1. ถ้าโฮลที่ต้องการฟรีเป็นโฮลตัวแรกในลิสต์แล้ว ให้เซตโฮลแรกของลิสต์ไปที่โฮลถัดไปแทน
2. ถ้าไม่ใช่โฮลแรกของลิสต์แล้ว ให้โฮลตัวก่อนชี้ไปที่โฮลถัดไปที่ชี้โดยโฮลนี้
3. ใส่โฮลนี้ไว้ในฟรีลิสต์ โดยให้โฮลถัดไปชี้ไปที่ฟรีลิสต์แล้วจึงให้ฟรีลิสต์ชี้มาที่โฮลนี้

ผู้เรียกใช้ alloc_mem, merge

ไฟล์ mm/alloc.c

ฟังก์ชัน `merge (hp)`

อินพุท
```
struct hole *hp;          /* พอยน์เตอร์ไปที่โฮลตัวก่อนที่จะนำมารวม */
```

เอาท์พุท ไม่มี

หน้าที่ พยายามรวมโฮลที่อยู่ติดกันในลิสต์ ที่มีแอดเดรสเริ่มต้นของโฮลต่อกับแอดเดรสสุดท้ายของโฮลตัวก่อน ให้เก็บอยู่ในโฮลตัวเดียวกัน

การทำงาน

1. ตรวจดูว่าโฮลถัดไปมีแอดเดรสเริ่มต้นต่อจากแอดเดรสสุดท้ายของโฮลนี้ ถ้าใช่ ให้ปรับขนาดของโฮลใหม่ให้มีขนาดเท่ากับสองโฮลรวมกัน และลบสล็อตถัดไป
2. ทำเหมือนข้อ 1 แต่ให้เช็คกับโฮลถัดไปแทน

ผู้เรียกใช้ free_mem

ไฟล์ mm/alloc.c

ฟังก์ชัน max_hole ()

อินพุท ไม่มี

เอาท์พุท ขนาดของโฮลที่ใหญ่ที่สุดที่อยู่ในโฮลลิสต์

หน้าที่ คำนวณหาบล็อคหน่วยความจำที่ใหญ่ที่สุดที่ว่างอยู่

การทำงาน

1. วนลูปตรวจดูขนาดของโฮลทั้งหมดในโฮลลิสต์
2. หาโฮลที่มีขนาดใหญ่ที่สุด

ผู้เรียกใช้ ทั่วๆไป

ไฟล์ mm/alloc.c

ฟังก์ชัน mem_init (clicks)

อินพุท phys_clicks clicks; /* ขนาดหน่วยความจำที่มีอยู่ */

เอาท์พุท ไม่มี

หน้าที่ ตั้งค่าเริ่มต้นของส่วนจัดการกับหน่วยความจำเช่น โฮลลิสต์ ฟรีลิสต์

การทำงาน

1. ให้โฮลแรกมีหมายเลขคลิกเริ่มต้นเป็นศูนย์ และมีขนาดเท่ากับ clicks ซึ่งเป็นขนาดหน่วยความจำที่ใหญ่ที่สุดที่นำมาใช้ได้ และโฮลถัดไปให้ชี้ไปที่ NIL

2. สำหรับโอลที่เหลือ สร้างลิงค์ของฟรีลิสต์ชี้ต่อๆกันไป

ผู้เรียกใช้ mm_init

ฟังก์ชันการทำงานต่าง ๆ ในระบบไฟล์

ไฟล์ fs/main.c

ฟังก์ชัน main ()

อินพุท ไม่มี

เอาท์พุท ไม่มี

หน้าที่การทำงาน เป็นโปรแกรมหลักของระบบไฟล์ ในการให้บริการแก่ผู้เรียกใช้ตามข่าวสารที่ได้รับ

1. เรียกใช้ fs_init ในการตั้งค่าต่างๆของโปรเซส
2. วนลูปไม่มีที่สิ้นสุด
3. เรียกใช้ get_work ในการดอยรับข่าวสารจากโปรเซสอื่น
4. ตรวจสอบดูหมายเลขฟังก์ชันที่จะเรียกใช้ว่าถูกต้องหรือไม่ ถ้าไม่ถูกต้องให้ส่งค่า E_BAD_CALL กลับ
5. เรียกใช้ฟังก์ชันที่ต้องการ
6. ถ้าตัวแปร dont_reply ไม่เท่ากับศูนย์ ให้กลับไปวนลูปในข้อ 2
7. ส่งข่าวสารตอบกลับ
8. ถ้า inode ถัดไปมีค่าไม่เท่ากับ NIL_INODE ให้เรียกใช้ read_ahead เพื่ออ่านบล็อคถัดไป
9. กลับไปที่ข้อ 2

ผู้เรียกใช้ kernel

ไฟล์ fs/cache.c

ฟังก์ชัน struct buf *get_block (dev, block, only_search)

อินพุท
```
register dev_nr   dev;          /* ชนิดของบล็อก */
register block_nr block;        /* หมายเลขบล็อกที่ต้องการ */
int   only_search;              /* ถ้าเป็น NORMAL จะสามารถอ่าน-
                                   เขียนจากดิสค์ได้ แต่ถ้าเป็น NO_READ
                                   จะไม่อ่านข้อมูลจากดิสค์ */
```

เอาท์พุท บัฟเฟอร์บล็อกที่ต้องการ

หน้าที่ หาบล็อกที่ต้องการจากแคชขึ้นมาอ่านหรือเขียน

การทำงาน

1. คำนวนหาบล็อกที่ต้องการอยู่ในแฮชเชนอันไหน (hash chain)
2. ถ้า dev ไม่เป็น NO_dev ก็ให้ทำการวนหาบล็อคนั้นจนกว่าหมดคิวแฮชเชน
   - ถ้าค้นหาบล็อคนั้นเจอและเป็นชนิดของบล็อคเดียวกัน ก็ให้เซตค่าจำนวนบล็อคที่ใช้ (bufs_in_use) และบล็อคที่ใช้ (b_count) เพิ่มอีก 1
3. เมื่อถึงขั้นตอนนี้แสดงว่าไม่มีบล็อคที่ต้องการอยู่ในคิว ก็ให้นำบล็อคที่ถูกใช้น้อยสุดและไม่ถูกเใช้ในขณะนั้นมาใช้ โดยจะทำดังนี้
   - ตรวจสอบว่า ถ้า bufs_in_use เท่ากับ NR_BUFS แสดงว่าไม่มีบัฟเฟอร์ที่ว่าง ก็ให้ออกจากโปรแกรมโดยเรียกฟังก์ชัน panic
   - ค้นหาบล็อคที่ไม่ใช้ตั้งแต่ต้นคิว
   - ถ้าบัฟเฟอร์ถูกใช้หมด ก็ให้ออกจากโปรแกรมโดยเรียกฟังก์ชัน panic
4. เคลื่อนย้ายบล็อคที่ว่างออกจากแฮชเชน
5. ถ้าบัฟเฟอร์นั้นถูกเซตเป็น DIRTY ก็จะทำการเขียนบัฟเฟอร์นั้นลงดิสค์
6. ใส่ค่าพารามิเตอร์ต่างๆเช่น หมายเลขอุปกรณ์ (device number) หมายเลขบล็อค (block number) เป็นต้น ลงในบล็อค และนำกลับใส่ลงในแฮชเชน
7. ถ้าหากหมายเลขอุปกรณ์ไม่เป็น NO_DEV และ only_search เป็น NORMAL ก็จะทำการอ่านข้อมูลจากดิสค์ใส่ลงในบล็อคที่ต้องการ โดยเรียกฟังก์ชัน

rw_block

8. ส่งบล็อกที่ต้องการกลับไปยังตัวฟังก์ชันที่เรียกมา

ผู้เรียกใช้ FS

ไฟล์ fs/cache.c

ฟังก์ชัน put_block (bp, block_type)

อินพุท
```
register struct buf  *bp;    /* พอยน์เตอร์ชี้ไปยังบัฟเฟอร์ที่จะไม่ใช้ */
int   block_type;            /* ชนิดของบล็อค เช่น INODE_BLOCK,
                                DIRECTORY_BLOCK และอื่นๆ */
```

เอาท์พุท ไม่มี

หน้าที่ ยกเลิกบัฟเฟอร์ที่ไม่ใช้ ถ้าหากบัฟเฟอร์นั้นจะถูกใช้อย่างรวดเร็ว (ไม่ถูกเซตเป็น ONE_SHOT) บัฟเฟอร์นั้นก็จะถูกนำไปไว้ที่ท้ายคิว แต่ถ้าบัฟเฟอร์นั้นจะไม่ใช้อีกแล้วก็จะนำมาเก็บที่หัวคิว

การทำงาน

1. ตรวจสอบว่า ถ้า bp เป็น NIL_BUF ให้กลับไปโปรแกรมที่เรียกมา
2. นำบล็อคที่ต้องการจะยกเลิกออกจากคิว โดยจะทำ
   - ลดค่าตัวแปร b_count ลง 1 และถ้า b_count>0 แสดงว่ายังมีโปรเซสอื่นใช้อยู่ ก็ให้กลับไปโปรแกรมที่เรียกมา
   - ลดค่าตัวแปร bufs_in_use ลง 1
   - เอาบล็อคออกจากคิว
3. นำบล็อคนี้กลับเข้าคิว โดยจะตรวจสอบว่าบล็อคนี้เป็น ONE_SHOT หรือไม่ ถ้าเป็น แสดงว่าไม่ต้องการใช้บล็อกนั้นอีกแล้ว ก็ให้นำมาใส่ไว้หน้าคิว แต่ถ้าไม่ใช่ ONE_SHOT ก็ให้นำบล็อคนั้นใส่ไว้หลังคิว
4. ถ้าหากเป็นบล็อคที่สำคัญเช่น INODE_BLOCK, DIRECTORY_BLOCK จะต้องทำการเก็บลงดิสค์ทันที โดยเรียกฟังก์ชัน rw_block
5. ถ้า block_type เป็น ZUPER_BLOCK ก็ให้เปลี่ยนเป็น NO_DEV ด้วย

ผู้เรียกใช้ FS

ไฟล์ fs/cache.c

ฟังก์ชัน alloc_zone (dev, z)

อินพุท
```
dev_nr  dev;      /* หมายเลขอุปกรณ์ที่ทำการจอง */
zone_nr z;        /* หมายเลขที่จะจองโซนหรือบริเวณใกล้เคียง */
```

เอาท์พุท หมายเลขโซนที่จองได้

หน้าที่ ทำการจองโซนตามหมายเลขของอุปกรณ์

การทำงาน

1. หาซุปเปอร์บล็อกตามหมายเลขของอุปกรณ์ โดยเรียกใช้ฟังก์ชัน get_super
2. ทำการคำนวณหาจุดเริ่มต้นของโซน
3. ทำการจองโซน ณ. จุดเริ่มต้นของโซน หรือบริเวณใกล้เคียง โดยการเรียกฟังก์ชัน alloc_bit
4. ตรวจสอบดูว่า ถ้าไม่สามารถจองโซนได้ ก็ให้ทำการพิมพ์ค่าผิดพลาดออกมาและส่งค่า NO_ZONE กลับ
5. ส่งค่าหมายเลขโซนที่จองได้กลับ

ไฟล์ fs/cache.c

ฟังก์ชัน free_zone (dev, numb)

อินพุท
```
dev_nr  dev;      /* หมายเลขอุปกรณ์ที่ทำการจอง */
zone_nr numb;     /* หมายเลขที่จะยกเลิก */
```

เอาท์พุท ไม่มี

หน้าที่ ทำการยกเลิกโซนตามหมายเลขของอุปกรณ์

การทำงาน

1. ถ้า numb เป็น NO_ZONE ให้กลับไปยังฟังก์ชันที่เรียกมา
2. หาซุปเปอร์บล็อกตามหมายเลขของอุปกรณ์ โดยเรียกใช้ฟังก์ชัน get_super

3. ทำการยกเลิกหมายเลขโซนนั้น

ไฟล์ fs/cache.c

ฟังก์ชัน rw_block (bp, rw_flag)

อินพุท
```
register struct buf  *bp;    /* พอยน์เตอร์ชี้ไปยังบัฟเฟอร์ */
int   rw_flag;               /* สถานะบอกว่าอ่านหรือเขียน */
```

เอาท์พุท ไม่มี

หน้าที่ ทำการอ่าน-เขียนลงบนดิสค์ ถ้าหากเกิดข้อผิดพลาดก็จะพิมพ์ข้อผิดพลาดออกมา แต่จะไม่ส่งค่ากลับไปยังฟังก์ชันที่เรียก

การทำงาน

1. ถ้าหากหมายเลขอุปกรณ์ของ bp ไม่เท่ากับ NO_DEV แล้ว จะทำดังนี้
   - หาตำแหน่งที่จะอ่านหรือเขียน
   - ทำการเรียกฟังก์ชัน dev_io เพื่อใช้ในการอ่านหรือเขียน
   - ถ้ามีข้อผิดพลาดก็จะพิมพ์ข้อผิดพลาดออกมา
2. เซตค่า b_dirt ของ bp เป็น CLEAN

ผู้เรียกใช้ get_block, put_block

ไฟล์ fs/cache.c

ฟังก์ชัน invalidate (device)

อินพุท
```
dev_nr  device;      /* หมายเลขอุปกรณ์ที่ทำการลบออก */
```

เอาท์พุท ไม่มี

หน้าที่ ทำการลบบล็อคที่มีหมายเลขอุปกรณ์ device ออกจากแคช

การทำงาน จะทำการวนลูปตามจำนวนของบัฟเฟอร์ และจะเซตค่าบัฟเฟอร์ที่มีหมายเลข device ให้เป็น NO_DEV

ผู้เรียกใช้ FS

ภาคที่ ๒

บทที่ 4

หลักการพัฒนาระบบภาษาไทยบนมินิกซ์

หลังจากได้ศึกษาถึงหลักการในการออกแบบและสร้างระบบปฏิบัติการมินิกซ์แล้ว ขั้นตอนต่อไปจะเป็นการพัฒนามินิกซ์ ให้เป็นระบบปฏิบัติการที่มีความสามารถในการรับตัวอักษรภาษาไทยและสามารถแสดงผลตัวอักษรภาษาไทย/อังกฤษได้ โดยไม่อาศัยฮาร์ดแวร์เพิ่มเติม โดยมีแนวทางในการพัฒนาเป็นส่วนๆดังนี้

- การแสดงผลทางจอภาพของมินิกซ์
- โครงสร้างข้อมูลในการเก็บตัวอักษรภาษาไทย/อังกฤษบนจอภาพ
- การแสดงผลกราฟฟิคเคอร์เซอร์
- การแม็ปแป้นพิมพ์ภาษาไทย
- การรับอินพุทภาษาไทย/อังกฤษ
- ฟอนต์ที่ใช้ในการแสดงผลระบบภาษาไทยมินิกซ์

ในบทนี้จะกล่าวถึงหลักการที่ใช้ในการออกแบบระบบภาษาไทยในส่วนต่างๆ โดยจะกล่าวถึงลักษณะของระบบภาษาไทยเป็นอันแรก เพื่อให้ทราบถึงลักษณะคร่าวๆของระบบปฏิบัติการมินิกซ์ภาษาไทย และลักษณะการแก้ไขปรับปรุงระบบเดิม

ลักษณะของระบบภาษาไทยบนระบบปฏิบัติการมินิกซ์

ลักษณะของระบบปฏิบัติการมินิกซ์ภาษาไทย จะมีคุณสมบัติและแนวทางในการแก้ไขดังต่อไปนี้

- สามารถแสดงผลทางตัวอักษรภาษาไทย/อังกฤษได้ ในที่นี้จะทำบนกราฟฟิกโหมดของจอภาพสีแบบอีจีเอ/วีจีเอ และจอภาพเบียวที่ใช้การ์ดเฮอร์คิวลิส โดยใช้ความละเอียด 640x350 จุดในการแสดงผล และแก้ไขทาล์กเทอร์มินัลในส่วนของเคอร์นัลให้มาเรียกใช้รูทีนในการแสดงผลทางกราฟฟิคโหมด
- สามารถแสดงผลเป็นระบบภาษาไทย/อังกฤษแบบไม่จัดระดับ 25 บรรทัด และใน

โหมดจัดระดับตัวอักษรจะสามารถแสดงผลได้ 17 บรรทัด และทั้งสองโหมดสามารถแสดงตัวอักษรได้ 80 ตัวอักษรต่อหนึ่งบรรทัด โดยได้สร้างฟอนต์ในการแสดงผลเป็นสองชุดสำหรับแต่ละโหมด และในแต่ละโหมดยังสามารถแสดงตัวอักษรได้อีก 2 รูปแบบ นอกจากนั้นโปรแกรมของผู้ใช้ยังสามารถติดตั้งฟอนต์เพิ่มได้อีก

- สามารถแสดงผลตัวอักษรได้หลายๆฟอนต์พร้อม ๆ กัน (multifont) และยังสามารถทำคำสั่งที่เกี่ยวกับกราฟฟิคได้ เช่น การลากเส้น การเขียนจุด เป็นต้น

- เพิ่มโปรเซสที่เกี่ยวกับระบบภาษาไทย เพื่อใช้บริการแก่ผู้ใช้โดยจัดทำเป็นซิสเต็มคอลล์ เช่น การอ่านตัวอักษรบนหน้าจอ การเปลี่ยนโหมดภาษาไทยแบบจัดและไม่จัดระดับ เป็นต้น

- เพิ่มซอฟท์แวร์เคอร์เซอร์ ซึ่งจะทำการกระพริบทุกๆครึ่งวินาที เพื่อใช้ในการบอกตำแหน่งของเคอร์เซอร์

- เพิ่มโปรแกรมประยุกต์ใช้งานบนมินิกซ์ซึ่งเป็นเชลล์ในการจัดการกับระบบ เช่น การก็อปปี้ไฟล์ การลบไฟล์ การแสดงไฟล์ การสร้างและลบไดเรกทอรี และการจัดการกับดีไวซ์ต่างๆโดยมีการใช้งานเป็นแบบเมนูพูลดาวน์ (Pull down menu system) และโปรแกรมประยุกต์ใช้งานอื่นๆ เช่น โปรแกรมอิดิตเตอร์ภาษาไทยแบบแก้ไขทีละบรรทัด โปรแกรมอ่านไฟล์ภาษาไทยมาแสดงยังจอภาพแบบหน้าต่าง (window) โปรแกรมเซตสีตัวอักษรและสีพื้นตัวอักษร เป็นต้น

- มีปุ่มคีย์ในการเลือกโหมดภาษาไทย/อังกฤษ ชุดของฟอนต์ที่ใช้ในการแสดงผลและลักษณะการแสดงผล

<u>หลักการและโครงสร้างในการออกแบบระบบภาษาไทยมินิกซ์</u>

การแสดงผลทางจอภาพของมินิกซ์

เนื่องจากมินิกซ์มีการแสดงผลออกทางจอภาพเป็นแบบกราฟฟิคโหมด ดังนั้นการแสดงผลภาษาไทยและอังกฤษของมินิกซ์ สามารถทำได้โดยการรวมส่วนฟอนต์รหัสแอสกี้ภาษาไทย/อังกฤษ ซึ่งเก็บเป็นแบบรหัสสมอ. เข้าเป็นส่วนหนึ่งของโอเอสมินิกซ์ การแสดงผลตัวอักษร

แต่ละตัวก็จะไปดึงเอาฟอนต์ของตัวอักษรนั้นๆมาแสดงผลตามตำแหน่งที่ต้องการ

การแสดงผลตัวอักษรบนมินิกซ์ จะแบ่งออกเป็น 2 โหมดด้วยกัน คือ

1. โหมดภาษาไทย/อังกฤษไม่จัดระดับ 25 บรรทัด เป็นโหมดที่มีคุณสมบัติเหมือนมินิกซ์เวอร์ชันที่เป็นเท็กซ์โหมดทุกประการ เพื่อคงไว้ซึ่งความทัดเทียม (compatibility) และการถ่ายเทโปรแกรม (portability) โดยซอฟท์แวร์ทุกตัวที่ทำงานบนมินิกซ์เวอร์ชันเดิมจะสามารถทำงานบนมินิกซ์กราฟฟิคโหมดนี้ได้ โดยไม่ต้องแก้ไขโปรแกรมส่วนใดๆ การแสดงผลภาษาไทยจะไม่มีการจัดระดับของสระบน,สระล่าง และวรรณยุกต์ เพราะมองตัวอักษรทุกตัวเป็นฟอนต์ปกติเหมือนกัน

2. โหมดภาษาไทย/อังกฤษจัดระดับ 17 บรรทัด เป็นโหมดที่มีความสามารถในการแสดงผลภาษาไทยแบบจัดระดับได้ มีการวางตำแหน่งสระบน, สระล่าง และวรรณยุกต์อย่างถูกต้องโดยอัตโนมัติ ตัวอักษรที่รับจากแป้นพิมพ์จะนำมาตรวจจัดระดับตามตำแหน่งที่ควรจะเป็น และการลบตัวอักษรจะลบเรียงลำดับย้อนกลับ เช่น คำว่า ปี่ เมื่อลบตัวอักษรทีละตัวจะเหลือเป็นปิ, ป ตามลำดับ ซึ่งหน้าที่ต่างๆเหล่านี้ โอเอสจะเป็นตัวจัดการทั้งสิ้น

สำหรับคอนโซล สามารถเปลี่ยนโหมดการแสดงผลทั้งสองโหมดกลับไปกลับมาได้ โดยการกดปุ่ม F8 หรืออาจจะเรียกใช้ผ่านโปรแกรมคอมมานด์ไลน์ก็ได้

โครงสร้างข้อมูลของจอภาพ

เนื่องจากการจัดระดับตัวอักษรและการลบตัวอักษรตามลำดับนั้น โอเอสจะต้องทราบถึงตัวอักษรที่อยู่บนจอภาพในขณะนั้น เพื่อที่จะนำไปประมวลผลให้ถูกต้อง ดังนั้นโอเอสจะต้องมีบัฟเฟอร์ที่ใช้สำหรับเก็บตัวอักษรที่ปรากฏอยู่บนจอภาพด้วย โดยมีการกำหนดโครงสร้างข้อมูลดังนี้

```
#define NR_ROW      25          /* number of row */
#define NR_COL      80          /* number of column */

/* define structure of visual buffer (video ram) */
```

```
struct VRAM {                       /* visual video ram */
        unsigned char ascii;        /* ascii of character */
        unsigned char extended;     /* extended byte */
        unsigned char font;         /* font number */
        unsigned char attr;         /* attribute or color */
} vvram[NR_ROW][NR_COL];            /* entire screen = NR_COL*NR_ROW */
```

ในหนึ่งหน้าจอภาพจะประกอบไปด้วยตัวอักษร 80 คอลัมน์ จำนวน 17 หรือ 25 บรรทัด โดยข้อมูลของตัวอักษรแถวที่ i คอลัมน์ที่ j จะถูกเก็บไว้ในบัฟเฟอร์ vvram[i][j] โดยแต่ละตัวอักษรจะเก็บข้อมูลต่างๆดังนี้

1. รหัสแอสกี้ เป็นรหัสของตัวอักษรนั้นๆ ในกรณีเป็นตัวอักษรภาษาอังกฤษ, ตัวอักษรพิเศษ เช่น กรอบ สัญลักษณ์ต่างๆ, และตัวพยัญชนะภาษาไทย จะเก็บไว้ในไบต์นี้

2. รหัสขยาย เป็นรหัสที่ใช้เก็บอักขระภาษาไทยที่มิได้อยู่ในระดับเดียวกับตัวอักษรปกติโดยแบ่งออกเป็นแต่ละบิต ดังรูป 4.1 และเนื่องจากรหัสขยายเก็บสระและวรรณยุกต์ด้วยรูปแบบภายใน ดังนั้นโอเอสจะต้องมีฟังก์ชันที่จะทำการแปลงรหัสเหล่านี้ออกเป็นรหัสแอสกี้ที่สอดคล้องกับรหัสนั้นด้วย ซึ่งสามารถทำได้โดยการเปิดตารางแปลงจากรหัสภายในเป็นรหัสสมอ.

3. หมายเลขฟอนต์ ในหน้าจอสามารถแสดงตัวอักษรได้หลายๆฟอนต์ด้วยกัน โดยจะเก็บหมายเลขฟอนต์ของแต่ละตัวอักษรที่ไบต์นี้

4. แอตริบิวท์ เป็นรหัสที่เก็บลักษณะของสีของตัวอักษรและสีพื้นของตัวอักษรนั้น โดยสี่บิตบนจะเก็บสีของพื้นตัวอักษร และสี่บิตล่างจะเก็บสีของตัวอักษรนั้นๆ

บิตที่	7	6	5	4	3	2	1	0

วรรณยุกต์
000 ว่าง
001 ไม้เอก
010 ไม้โท
011 ไม้ตรี
100 ไม้จัตวา
101 ไม้ทัณฑฆาต

สระบน
000 ว่าง
001 ไม้ไต่คู้
010 ไม้หันอากาศ
011 สระอิ
100 สระอี
101 สระอึ
110 สระอือ
111 นฤคหิต

สระล่าง
000 ว่าง
001 สระอุ
010 สระอู
011 พินทุ

รูป 4.1 แสดงการเก็บรูปแบบภายในของตัวอักษรขยาย

การแสดงผลเคอร์เซอร์

เนื่องจากการแสดงผลทางกราฟฟิคโหมดไม่มีฮาร์ดแวร์ในการแสดงเคอร์เซอร์ ดังนั้นการแสดงเคอร์เซอร์ จะกระทำโดยซอฟท์แวร์ โดยส่วนที่จัดการกับเคอร์เซอร์จะอยู่ในทาสค์คล็อค (clock task) ซึ่งจะสั่งให้มีการแสดงเคอร์เซอร์ทุกๆ 0.5 วินาที โดยจะปรับตำแหน่งของเคอร์เซอร์ให้ถูกต้องตลอดเวลา

ในการเขียนเคอร์เซอร์ลงบนจอภาพนั้น จะทำโดยการเขียนข้อมูลแบบ XOR กับข้อมูลเดิมบนจอภาพ ดังนั้นในการลบเคอร์เซอร์ก็จะทำได้โดยการเขียนเคอร์เซอร์ไปที่ตำแหน่งนั้นอีกครั้ง ข้อมูลบนจอภาพก็จะปรากฏเป็นอย่างเดิม

ปัญหาที่ตามมาคือ ถ้าในระหว่างจอภาพแสดงเคอร์เซอร์อยู่นั้นมีการเขียนข้อมูลทับไป

ที่ตำแหน่งของเคอร์เซอร์ ก็จะมีผลทำให้การลบเคอร์เซอร์ครั้งต่อไปผิด ภาพของเคอร์เซอร์บางส่วนที่ลบไม่หมดจะปรากฏเป็นขยะเหลืออยู่บนจอภาพ ดังนั้นทางแก้จึงอยู่ที่ว่าก่อนจะมีการแก้ไขข้อมูลใดๆบนจอ จะต้องลบเคอร์เซอร์ออกก่อนทุกครั้งและก็จะเซตแฟลกไว้เพื่อป้องกันมิให้ทาสค์คล็อควาดเคอร์เซอร์ที่เวลา 0.5 วินาทีถัดไป จนกว่าจะมีการแก้ไขหน้าจอเสร็จ แล้วจึงเคลียร์แฟลกนั้นทิ้ง

การแม็ปคีย์บอร์ดภาษาไทย

ในระบบปฏิบัติการมินิกซ์นั้น การแปลงอินพุทจากคีย์บอร์ด ไปเป็นรหัสแอสกี้ เพื่อนำไปประมวลผล หรือแสดงผลนั้น จะใช้การเก็บตารางซึ่งจะเป็นการแม็ปจากสแกนโค้ด ซึ่งได้มาจากคีย์บอร์ด ไปเป็นรหัสแอสกี้ โดยตารางดังกล่าวจะอยู่ในซอร์สโค้ดไฟล์ ttymaps.h โดยจะเก็บตารางไว้หลายชุดด้วยกัน ซึ่งแต่ละชุดก็จะใช้งานกับคีย์บอร์ดแต่ละชนิด รวมทั้งเก็บตารางของการกดคีย์บอร์ดปกติ และการกดคีย์บอร์ดพร้อมกับปุ่ม SHIFT แยกกันด้วย

ดังนั้น เมื่อต้องการปรับปรุงให้มินิกซ์สามารถรับอินพุทจากคีย์บอร์ดเป็นภาษาไทยได้ก็สามารถทำได้โดยง่าย กล่าวคือ ในที่นี้ได้ทำการเพิ่มตารางอีก 2 ตาราง เพื่อใช้ในการแม็ปอินพุทจากคีย์บอร์ดธรรมดา และอินพุทที่มีการกด SHIFT ให้เป็นรหัสแอสกี้ของภาษาไทย

การรับอินพุทภาษาไทยและภาษาอังกฤษ

เมื่อสามารถทำให้มินิกซ์แม็ปคีย์บอร์ดภาษาไทยได้แล้ว ขั้นตอนต่อไป คือการทำให้สามารถรับอินพุทได้ทั้งสองภาษา ในที่นี้ได้ทำการกำหนดตัวแปรเพื่อใช้บอกสถานะของอินพุท ว่าในขณะนั้นๆ อินพุทเป็นภาษาใด โดยมีชื่อว่า input_mode อยู่ในไฟล์ glo.h โดยตัวแปรนี้ จะมีค่าได้เพียงสองค่าเท่านั้น คือ IN_THAI หรือ IN_ENG และได้ทำการแก้ไขฟังก์ชัน func_key ซึ่งอยู่ในไฟล์ console.c เพื่อว่าเมื่อทำการกดปุ่ม F10 จะเป็นการสวิทช์ระหว่างภาษาไทย และภาษาอังกฤษ เช่น ถ้าเดิมอินพุทเป็นภาษาไทยอยู่ เมื่อทำการกดปุ่ม F10 ก็จะทำให้อินพุทกลายเป็นภาษาอังกฤษ แต่ถ้าเดิมอินพุทเป็นภาษาอังกฤษ เมื่อกดปุ่ม F10 อินพุทก็จะกลายเป็นภาษา

ไทย โดยการทำงานจริง ก็เพียงแค่เปลี่ยนค่าตัวแปร input_mode เท่านั้น (อนึ่ง การที่ใช้ปุ่ม F10 เป็นปุ่มสวิทช์โหมดนั้น เป็นเพียงการทดลอง ในอนาคต อาจมีการเปลี่ยนแปลงได้ตามความเหมาะสม)

เมื่อสามารถทำการสวิทช์อินพุทโหมดได้แล้ว โดยการเซ็ตค่าไว้ในตัวแปร input_mode ในการอินพุทสองภาษา ก็สามารถทำได้โดยการตรวจสอบว่าขณะนั้นอินพุทโหมดเป็นภาษาใด แล้วให้ใช้ตารางแม็ปของภาษานั้น ๆ โดยในที่นี้ ได้ทำการแก้ไขฟังก์ชัน make_break ซึ่งอยู่ในไฟล์ tty.c ซึ่งทำหน้าที่แม็ปสแกนโค้ดจากคีย์บอร์ด ไปเป็นรหัสแอสกี้ โดยในการแก้ไข จะทำให้ต้องตรวจสอบค่าของ input_mode ก่อนว่าเป็น IN_THAI หรือ IN_ENG ถ้าเป็น IN_ENG การแม็ป ก็จะใช้ตารางของภาษาอังกฤษ แต่ถ้าเป็น IN_THAI ก็จะใช้ตารางของภาษาไทยที่ได้ทำการเพิ่มไป

ฟอนต์ที่ใช้ในระบบภาษาไทยมินิกซ์

ฟอนต์ในการแสดงผลบนระบบภาษาไทยมินิกซ์ จะแบ่งเป็น 2 ขนาดด้วยกัน คือขนาด 8x14 จุด ซึ่งสามารถแสดงผลทางจอภาพได้ 80x25 บรรทัด และขนาด 8x20 จุด ซึ่งสามารถแสดงผลทางจอภาพได้ 17 บรรทัด โดยในแต่ละขนาดจะมีลักษณะตัวอักษร 2 ฟอนต์ๆละ 256 ตัวอักษร ดังนั้นขนาดของฟอนต์ทั้งหมดจะคำนวณได้ดังนี้

ขนาดของฟอนต์ = ขนาดของฟอนต์ 8x14 + ขนาดของฟอนต์ 8x20

= (ขนาดฟอนต์ 8x14 ชุดที่ 1 + ขนาดฟอนต์ 8x14 ชุดที่ 2) +
(ขนาดฟอนต์ 8x20 ชุดที่ 1 + ขนาดฟอนต์ 8x20 ชุดที่ 2)

= (2 ชุด x 256 ตัวอักษร x 14 สูง x 8 กว้าง) +
(2 ชุด x 256 ตัวอักษร x 20 สูง x 8 กว้าง) บิต

= 7168 ไบต์ + 10,240 ไบต์

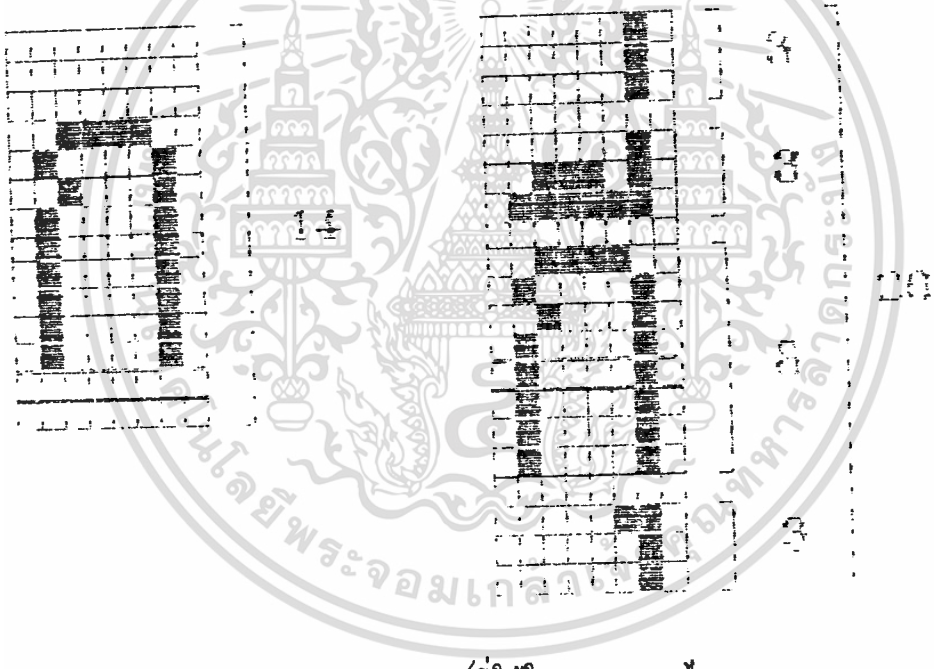
= 17,408 ไบต์

ฟอนต์ชุดนี้จะถูกสร้างรวมลงในระบบภาษาไทยมินิกซ์ ซึ่งจะทำให้ขนาดของระบบใหญ่ขึ้นอีก 17 กิโลไบต์ โดยไม่รวมถึงโค้ดที่จัดการกับระบบภาษาไทย ด้วยเหตุนี้ในการออกแบบจึง

เลือกที่จะเก็บฟอนต์เป็น 2 ชุดเท่านั้น เพื่อประหยัดหน่วยความจำที่ใช้ในการเก็บฟอนต์

ลักษณะของฟอนต์จะต้องเก็บเป็นแบบบิตแมป (bit map) เป็นตารางรหัสแอสกี้ภาษาไทยสมอ. โดยฟอนต์แต่ละชุดมีวิธีการในการรวมกันเป็นฟอนต์ไฟล์เดียวและสร้างรวมลงในแผ่นบู๊ตมินิกซ์ ซึ่งจะอธิบายไว้ในบทที่ 6 อย่างละเอียด

สำหรับการเปลี่ยนฟอนต์ สามารถกระทำได้โดยการกดปุ่ม F9 ซึ่งจะสลับฟอนต์ไปมาระหว่างชุดที่หนึ่งและชุดที่สอง ถ้าผู้ใช้ต้องการแสดงผลให้มากกว่าหนึ่งฟอนต์สามารถทำได้โดยการเรียกชิสเต็มคอลล์ติดตั้งฟอนต์ โดยรายละเอียดฟังก์ชันที่ใช้ในการติดตั้งฟอนต์ได้ในภาคผนวก



รูป 4.2 แสดงลักษณะของฟอนต์ที่ใช้ในระบบภาษาไทย 2 ขนาด

บทที่ 5

วิธีการสร้างระบบภาษาไทยบนมินิกซ์

ในบทนี้จะกล่าวถึงรายละเอียดในการพัฒนาแก้ไขระบบปฏิบัติการมินิกซ์เดิม ให้มีความสามารถในการแสดงระบบภาษาไทย โดยจะแสดงรายละเอียดของไฟล์ที่ทำการแก้ไข แสดงการทำงานของโปรแกรมตามเหตุการณ์ (events) ต่างๆ การทำส่วนจัดการกราฟฟิค และการแก้ไขคอมมานด์เชลล์เพื่อให้รับข้อความภาษาไทย

ข้อกำหนดในการแก้ไขโปรแกรม

การพัฒนาระบบปฏิบัติการมินิกซ์ จะประกอบไปด้วยการแก้ไขไฟล์ต่างๆในระบบเดิม และการสร้างไฟล์เพิ่มเติมขึ้น ดังนั้นเพื่ออำนวยความสะดวกในการตรวจดูถึงจุดต่างๆที่ได้เพิ่มเติมหรือแก้ไข ซอร์สโค้ดของแต่ละไฟล์ที่มีการแก้ไขจะอยู่ในเงื่อนไขการดีฟายน์สัญญลักษณ์ THAI ดังนี้

```
#ifdef THAI
    /* ส่วนที่มีการแก้ไข */
    ....
#endif
```

ดังนั้นการคอมไพล์เพื่อที่จะสร้างระบบไฟล์ จะกระทำโดยการเรียกคอมไพเลอร์และนิยามสัญญลักษณ์ THAI ไว้ ดังตัวอย่าง

```
cc -c -D THAI main.c
```

หรือเรียกใช้โปรแกรม make ซึ่งโปรแกรมเมคจะไปอ่านเมคไฟล์ที่มีการแก้ไขโดยเพิ่มออปชัน -D THAI ไว้แล้ว ดังนั้นในการสร้างระบบส่วนต่างๆ เช่น เคอร์นัล, ส่วนจัดการหน่วยความจำ, ส่วนระบบไฟล์ สามารถทำได้โดยการไปที่ไดเรกทอรีนั้นๆ และเรียกโปรแกรมเมค ดังนี้

```
make
```

สำหรับเมคไฟล์ที่มีชื่อว่า makefile อยู่แล้ว หรือถ้าเมคไฟล์มีชื่อไฟล์อย่างอื่น ก็สามารถเมคได้โดยกำหนดชื่อเมคไฟล์ ดังนี้

```
make -f hercules.mak
```

รายละเอียดไฟล์ที่ใช้งาน

ไฟล์ที่ใช้งานในการพัฒนามินิกซ์ จะอยู่ในไดเร็กตอรีต่อไปนี้

1. h
2. kernel
3. mm
4. tools
5. tlibsrc

สำหรับไดเร็กตอรี fs ซึ่งเป็นส่วนระบบไฟล์นั้น ไม่มีการแก้ไขใด ๆ ทั้งสิ้น

สำหรับไฟล์ใด ที่มีการเปลี่ยนแปลงเพิ่มเติม จะแสดงชื่อไฟล์เป็นตัวอักษรเข้ม ส่วนไฟล์ที่สร้างขึ้นใหม่ จะแสดงชื่อไฟล์เป็นตัว*อักษรเอียง* ซึ่งไฟล์เหล่านี้ จะมีซอร์สโค้ดอยู่ในภาคผนวกของวิทยานิพนธ์เล่มนี้

1. <u>ไดเร็กตอรี h</u>

เป็นไดเร็กตอรีที่เก็บเฮดเดอร์ไฟล์ ที่ทั้งเคอร์นัล ส่วนจัดการหน่วยความจำ และส่วนระบบไฟล์ ใช้ร่วมกัน ประกอบด้วยไฟล์ต่าง ๆ ดังนี้

callnr.h	**com.h**	const.h
error.h	sgtty.h	signal.h
stat.h	**type.h**	

2. <u>ไดเร็กตอรี kernel</u>

เป็นไดเร็กตอรีที่เก็บซอร์สโค้ดของส่วนเคอร์นัล ประกอบด้วยเฮดเดอร์ไฟล์ดังนี้

const.h	**glo.h**	**proc.h**
thailib.h	**tty.h**	**ttymaps.h**
type.h		

ประกอบด้วยซอร์สโค้ดภาษาแอสเซมบลีคือ

grlib.s	*jgrlib.s*
klib88.s	mpx88.s

และประกอบด้วยซอร์สโค้ดภาษาซีดังนี้

at_wini.c	clock.c	console.c
dmp.c	floppy.c	main.c
memory.c	printer.c	proc.c
ps_wini.c	rs232.c	system.c
table.c	*thailib.c*	*thaisys.c*
tty.c	wini.c	xt_wini.c

สำหรับไฟล์ที่ใช้ในการสร้างเคอร์นัล คือไฟล์

makefile

3. ไดเร็กตอรี mm

เป็นไดเร็กตอรีที่เก็บไฟล์ต่าง ๆ ที่เป็นซอร์สโค้ดของส่วนจัดการหน่วยความจำ จะประกอบด้วยเฮดเดอร์ไฟล์ ดังนี้

const.h	glo.h	mproc.h
param.h	type.h	

และประกอบด้วยซอร์สโค้ดภาษาซี คือ

alloc.c	break.c	cmode.c
forkexit.c	getpid.c	getset.c
main.c	putc.c	signal.c
table.c	*thai.c*	utility

สำหรับไฟล์ที่ใช้ในการสร้างส่วนจัดการหน่วยความจำ คือไฟล์

makefile

4. ไดเร็กตอรี tools

เป็นไดเร็กตอริที่ใช้ในการสร้างแผ่นบูต ประกอบด้วยไฟล์ซอร์สโค้ดที่สำคัญคือ

bootblok.s build.c init.c

และประกอบด้วยไฟล์รูปแบบตัวอักษร ได้แก่ ไฟล์ที่ชื่อลงท้ายด้วย .fon และไฟล์ที่รวมรูปแบบตัวอักษรทั้งสี่ชุด ได้แก่ไฟล์ชื่อ font ซึ่งมีวิธีการสร้างกล่าวไว้ในบทที่ 6

5. ไดเร็กตอรี tlibsrc

เป็นไดเร็กตอริที่เก็บซอร์สโค้ดของ ไลบรารีที่เกี่ยวกับชิสเต็มคอลล์ที่เพิ่มขึ้นใหม่ ประกอบด้วยเฮดเดอร์ไฟล์ คือ

tlib.h

และซอร์สโค้ดภาษาซี ดังนี้

clrkey.c	*clrscr.c*	*dmctrl.c*
gdmode.c	*getcolor.c*	*getx.c*
gety.c	*gkmode.c*	*gotoxy.c*
kmctrl.c	*kpressed.c*	*line.c*
offcursor	*oncursor.c*	*scrdown.c*
scrup.c	*sdmode.c*	*setcolor.c*
skmode.c	*xyac.c*	*xyas.c*

นอกจากนี้ ยังมีซอร์สโค้ดของโปรแกรมทดสอบชิสเต็มคอลล์ด้วย คือ

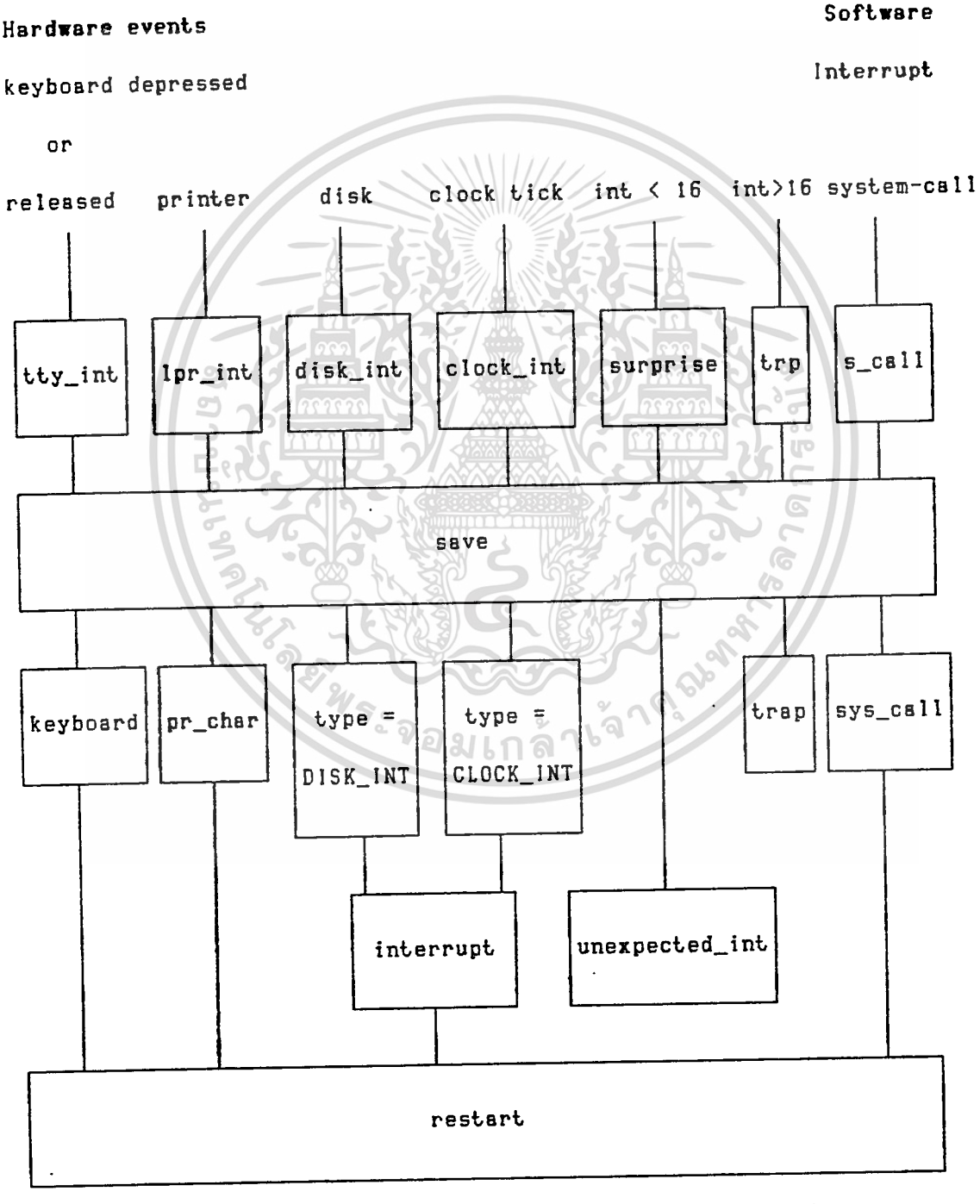
libtest.c

การไหลของชุดคำสั่งตามเหตุการณ์ต่างๆ

ระบบภาษาไทยมินิกซ์มีการจัดการกับเหตุการณ์ต่างๆ โดยอธิบายในรูปของการไหลของชุดคำสั่ง (code flow) ตามชื่อฟังก์ชันได้ตามเหตุการณ์ต่างๆดังนี้

1) ระบบอินเทอร์รัปต์ในมินิกซ์ เป็นการจัดการกับสัญญาณอินเตอร์รัพท์ทั้งทาง

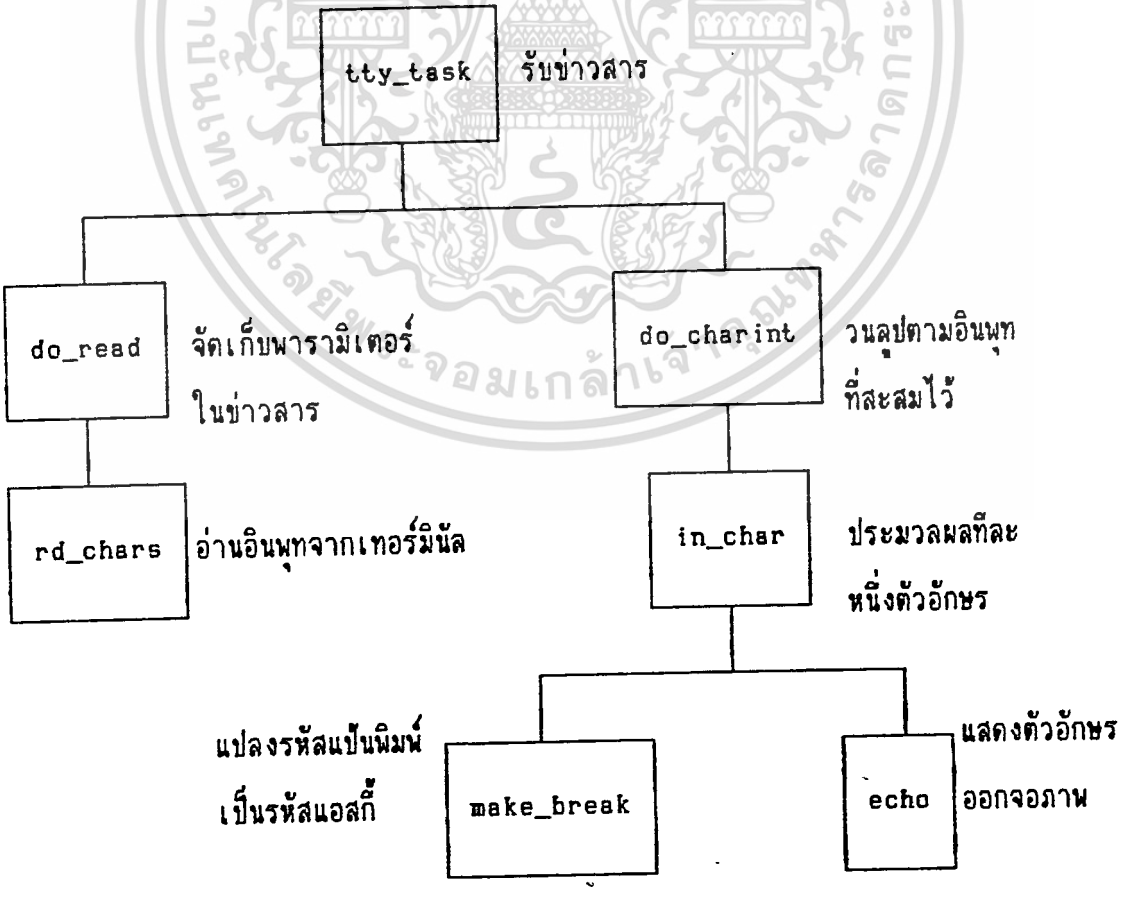
อาร์ดแวร์และซอฟท์แวร์ โดยส่วนจัดการกับสัญญาณอินเทอร์รัปต์ (interrupt servire routine) จะเป็นฟังก์ชันที่เขียนด้วยภาษาแอสเซมบลี้ และจะทำการแปลงสัญญาณอินเทอร์รัปต์ให้อยู่ในรูปของข่าวสาร (message) เพื่อส่งไปยังทาสค์ต่างๆที่มีหน้าที่ในการจัดการกับข่าวสารนั้นๆ โดยมีการแสดงการไหลของชุดคำสั่งไว้ในรูปที่ 5.1



รูป 5.1 แสดงการไหลของชุดคำสั่งเมื่อเกิดสัญญาณอินเทอร์รัปต์

2) ระบบการรับอินพุทจากเทอร์มินัล จะเกิดขึ้นเป็นสองส่วนด้วยกันคือ ส่วนรอรับอินพุทจากเทอร์มินัล และส่วนจัดการกับการกดหรือปล่อยแป้นพิมพ์ เนื่องจากระบบปฏิบัติการมินิกซ์ที่ทำงานบนเครื่องไอบีเอ็ม พีซี จะเกิดฮาร์ดแวร์อินเทอร์รัปต์เมื่อเกิดการกดหรือปล่อยปุ่มคีย์ใดๆดังนั้นส่วนจัดการต้องมีความสามารถในการจัดการการกด/ปล่อยคีย์ด้วย โดยทำการแมปตัวอักษรจากปุ่มคีย์ซึ่งเก็บเป็นรหัสสแกนโค้ดให้เป็นรหัสแอสกี้โค้ด โดยตรวจดูว่าในขณะนั้นอยู่ในโหมดการป้อนตัวอักษรภาษาไทยหรือภาษาอังกฤษ เพื่อนำไปเปิดตารางแมปตัวอักษรให้ถูกต้อง โดยตารางการแมปตัวอักษรนั้น จะเก็บอยู่ในไฟล์ ttymaps.h หลังจากนั้นก็จะทำการแสดงตัวอักษรนั้นออกทางเทอร์มินัลเอาท์พุทและจัดเก็บตัวอักษรไว้ในบัฟเฟอร์อีกด้วย

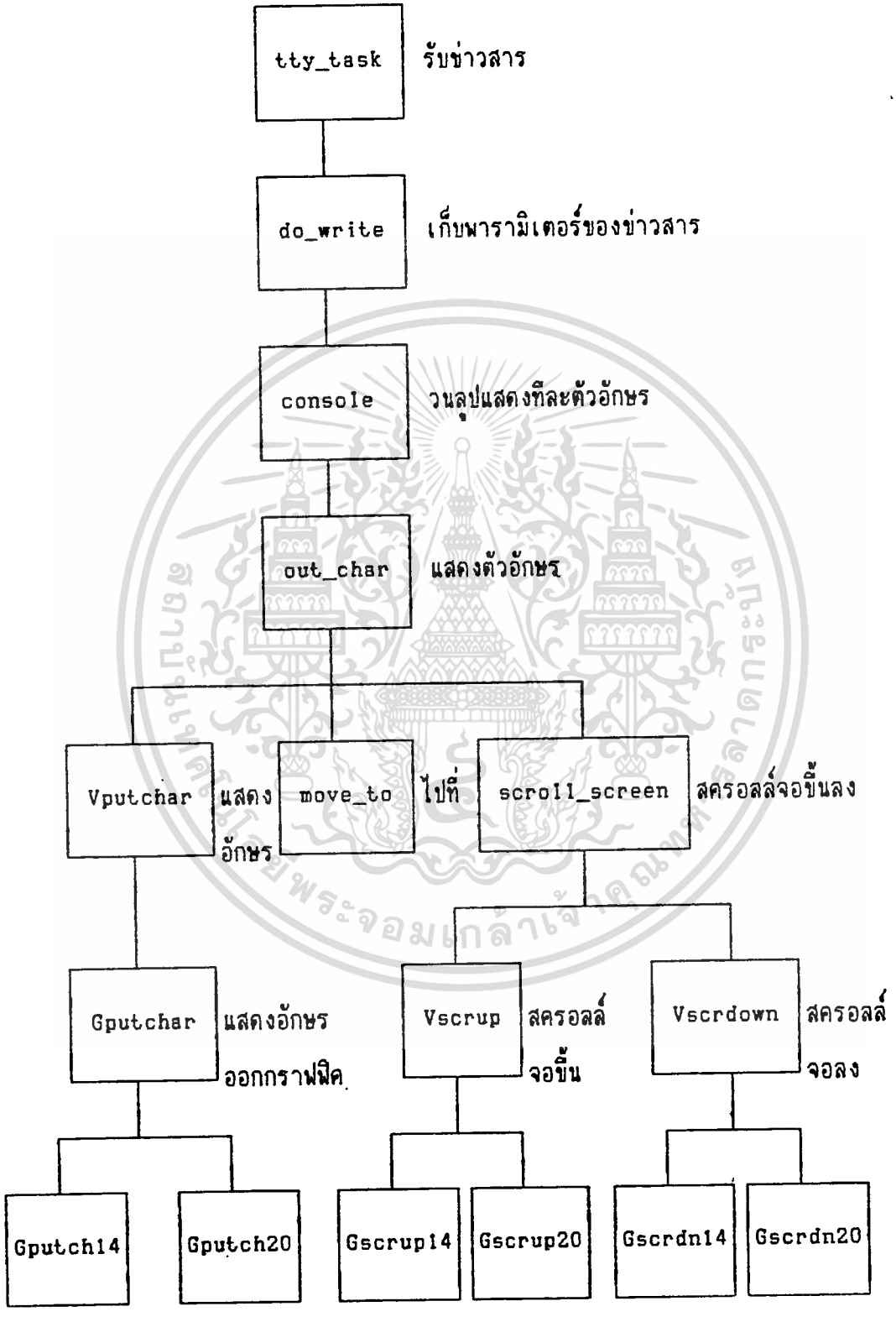
สำหรับส่วนการรอรับอินพุทจากเทอร์มินัล ก็จะตรวจดูว่ามีตัวอักษรอยู่ในบัฟเฟอร์พอเพียงกับที่ต้องการหรือไม่ ถ้าไม่พอก็จะทำการรอโดยการบล็อคตัวเองไว้ เมื่อตัวอักษรพอเพียงกับที่ต้องการก็จะลอกตัวอักษรชุดนั้นไปยังโปรเซสที่ต้องการอินพุทจากเทอร์มินัล ดังรูป 5.2



รูป 5.2 แสดงการไหลของชุดคำสั่งเมื่อมีการรับอินพุทจากเทอร์มินัล

3) การทำเทอร์มินัลเอาท์พุท การแสดงเอาท์พุทออกทางเทอร์มินัลของระบบภาษาไทยมินิกซ์จะไม่ใช้การฟลัช (flush) บัฟเฟอร์ลงบนวิดีโอแรม แต่จะใช้การแสดงทีละตัวอักษร เนื่องมาจากข้อจำกัดในการแสดงตัวอักษรออกทางกราฟฟิค นั่นคือ เมื่อมีการแสดงตัวอักษรลงบนจอภาพ ตัวอักษรแต่ละตัวจะถูกวาดให้เสร็จก่อนที่จะแสดงตัวอักษรถัดไป นอกจากนี้การทำเทอร์มินัลเอาท์พุทยังมีหน้าที่ในการจัดการกับตัวอักษรควบคุมต่างๆ เช่น ตัวอักษรควบคุมการขึ้นบรรทัดใหม่ การเลื่อนเคอร์เซอร์กลับ การกระโดดไปที่ตำแหน่งใดๆของจอภาพ เป็นต้น

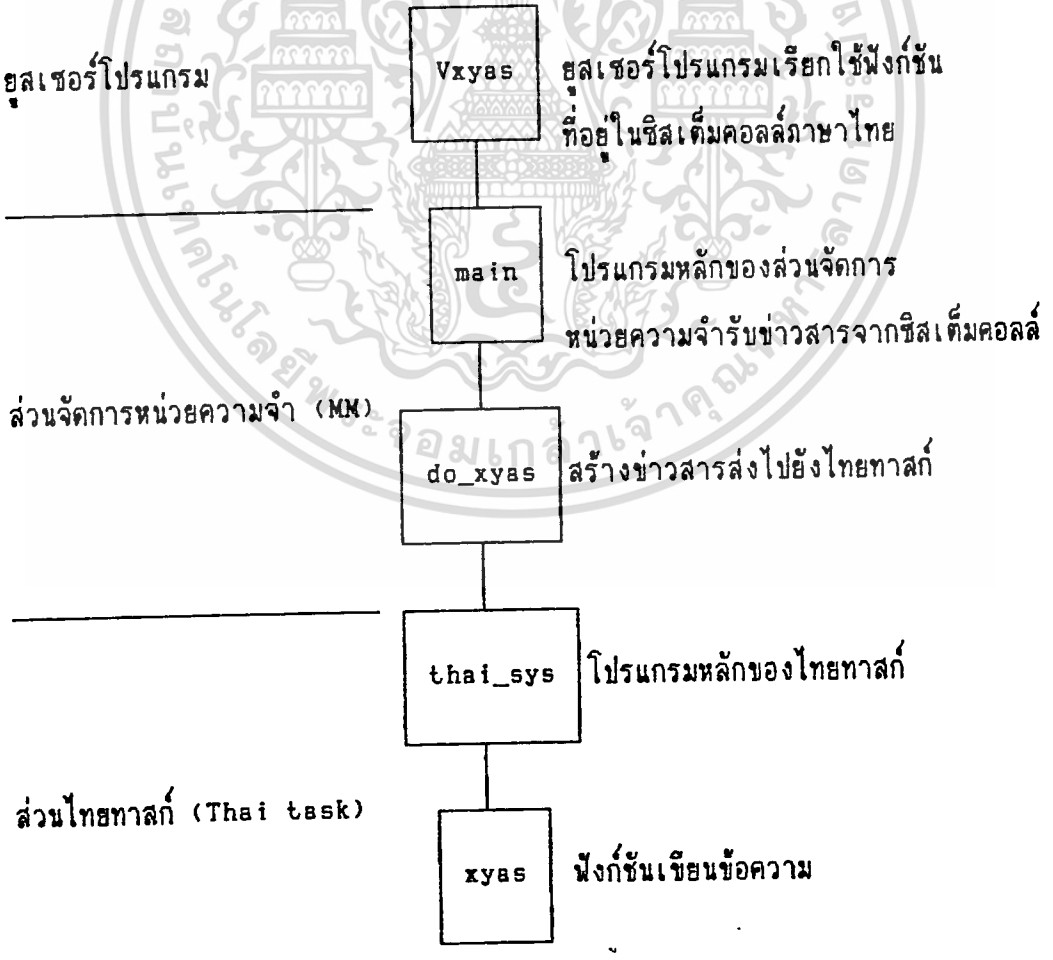
การทำเทอร์มินัลเอาท์พุทในระบบภาษาไทยมินิกซ์ จะเกี่ยวข้องกับการแสดงผลทางกราฟฟิคโดยตรง จึงมีผลมากในการจัดการกับระบบกราฟฟิค โดยฟังก์ชันพื้นฐานที่เกี่ยวข้องกับการแสดงผลทางกราฟฟิคจะเขียนด้วยภาษาแอสเซมบลี้ เช่น ฟังก์ชันในการเขียนตัวอักษร ฟังก์ชันในการสครอลล์จอภาพ เป็นต้น โดยจัดเก็บอยู่ในไฟล์ grlib.s และ tgrlib.s สำหรับจออีจีเอและวีจีเอ และไฟล์ jgrlib.s สำหรับจอเฮอร์คิวลิส และเนื่องจากโหมดในการแสดงผลภาษาไทยมีอยู่สองแบบ ดังนั้นในการแสดงผลจะต้องมีฟังก์ชันจัดการเรียกใช้ฟังก์ชันในการแสดงผลให้ถูกต้อง โดยฟังก์ชันเหล่านี้จะอยู่ในไฟล์ thailib.c การแสดงผลเอาท์พุทออกทางเทอร์มินัลสามารถแสดงอยู่ในรูปของการไหลของชุดคำสั่ง ได้ดังรูป 5.3



โหมด25บรรทัด โหมด17บรรทัด โหมด25บรรทัด โหมด17บรรทัด โหมด25บรรทัด โหมด17บรรทัด

รูป 5.3 แสดงการไหลของชุดคำสั่งเมื่อมีการแสดงผลเอาท์พุทออกทางเทอร์มินัล

4) การเรียกซิสเต็มคอลล์ภาษาไทย เมื่อมีการเรียกใช้ฟังก์ชันที่อยู่ในซิสเต็มคอลล์ภาษาไทย เช่นฟังก์ชัน Vxyas ที่ใช้ในการแสดงข้อความที่ตำแหน่งใดๆพร้อมกับเซตสีแอตริบิวต์นั้น การทำงานจะประกอบไปด้วยการเรียกไปยังฟังก์ชันที่อยู่ในซิสเต็มคอลล์ภาษาไทย ทำให้ฟังก์ชันนั้นสร้างข่าวสารและส่งไปยังส่วนจัดการหน่วยความจำ(MM)และบล็อคตัวเองจนกว่าส่วนจัดการหน่วยความจำจะตอบกลับ ในขณะเดียวกันส่วนจัดการหน่วยความจำก็จะตรวจความถูกต้องของข่าวสาร เช่น หมายเลขฟังก์ชัน ผู้เรียกใช้ เป็นต้น ถ้าไม่พบข้อผิดพลาด ส่วนจัดการหน่วยความจำก็จะสร้างข่าวสารส่งไปยังไทยทาสก์อีกทอดหนึ่ง และบล็อคตัวเองจนกว่าไทยทาสก์จะทำงานเสร็จ สำหรับไทยทาสก์จะอยู่ในไฟล์ thaisys.c ซึ่งจะทำหน้าที่ในการวนลูปอนันต์คอยรับข่าวสารและทำงานตามข่าวสารนั้น ซึ่งก็จะไปเรียกใช้ฟังก์ชันต่างๆทางกราฟฟิคที่จัดเก็บไว้ในไฟล์thailib.c นั่นเอง โดยยกตัวอย่างการแสดงรูปการไหลของชุดคำสั่ง Vxyac ได้ดังรูป 5.4



รูป 5.4 แสดงการไหลของชุดคำสั่งเมื่อมีการเรียกใช้ซิสเต็มคอลล์ภาษาไทย

ชิสเต็มคอลล์

หลังจากที่ทำการปรับปรุงมินิกซ์ให้สามารถทำงานกับภาษาไทยได้แล้ว ขั้นตอนต่อไปที่จะต้องทำก็คือ ทำอย่างไรให้ผู้เขียนโปรแกรมประยุกต์ใช้งาน สามารถเขียนโปรแกรมให้ใช้ความสามารถภายในของมินิกซ์ได้อย่างเต็มที่ ซึ่งก็คือการจัดการให้มินิกซ์มีชิสเต็มคอลล์นั่นเอง โดยผู้เขียนโปรแกรมสามารถเรียกใช้ชิสเต็มคอลล์เหล่านี้ได้โดยใช้ภาษาซี โดยจะเปรียบเสมือนชิสเต็มคอลล์เหล่านี้ เป็นชุดฟังก์ชันในภาษาซีที่เพิ่มขึ้นมา

การเรียกใช้ชิสเต็มคอลล์

ผู้เขียนโปรแกรมสามารถเรียกใช้ชิสเต็มคอลล์ได้ โดยจะต้องเรียกผ่านภาษาซี ซึ่งจะเรียกใช้เหมือนกับเป็นฟังก์ชันในภาษาซีนั่นเอง (ผู้อ่านสามารถพัฒนาชิสเต็มคอลล์ของมินิกซ์ให้สามารถถูกเรียกใช้ในภาษาอื่น ๆ ได้โดยไม่ยาก ถ้ามีคอมไพเลอร์ภาษานั้น ๆ ในมินิกซ์)

ชื่อของชิสเต็มคอลล์ที่สร้างเพิ่มขึ้นนี้ จะขึ้นต้นด้วยอักษรวี ('V') เช่น Vclrscr, Vxyac, Vline เป็นต้น โดยรายชื่อ และการใช้งานของชิสเต็มคอลล์แต่ละตัว ได้อธิบายไว้อย่างละเอียดในภาคผนวกของวิทยานิพนธ์เล่มนี้

เมื่อผู้เขียนทำการคอมไพล์โปรแกรมที่มีการเรียกใช้ชิสเต็มคอลล์เหล่านี้ จะต้องทำการลิงค์กับไฟล์ tlib.a ด้วย มิฉะนั้นโปรแกรม asld จะแจ้งข้อความว่า Unresolved reference _Vxxxx เช่น ถ้าในโปรแกรมมีการเรียกใช้ชิสเต็มคอลล์ Vclrscr เมื่อทำคอมไพล์โปรแกรม asld จะแจ้งข้อความ Unresolved reference _Vclrscr ซึ่งหมายความว่าโปรแกรมลิงค์ คือ asld หาฟังก์ชัน Vclrscr ไม่พบ ซึ่งจริง ๆ แล้วฟังก์ชันนี้ อยู่ในไฟล์ tlib.a นั่นเอง ดังนั้นทางแก้ก็คือ ต้องใส่ชื่อไฟล์ tlib.a ตามหลังชื่อโปรแกรม ในตอนคอมไพล์ด้วย กล่าวคือ

```
$ cc [options] xxx.c yyy.c zzz.c (...) tlib.a
```

การทำงานภายในของชิสเต็มคอลล์

เนื่องจากส่วนที่ได้ทำการปรับปรุงให้มีความสามารถเพิ่มขึ้น ก็คือ ส่วนเคอร์นัล ดังนั้น ส่วนที่จะทำงานจริงตามชิสเต็มคอลล์ก็คือ ส่วนเคอร์นัลนั่นเอง และจากที่ได้กล่าวมาแล้วในบทต้น ๆ ว่า มินิกซ์มีการติดต่อระหว่างโปรเซส โดยใช้ระบบข่าวสาร ดังนั้นการที่จะทำให้โปรแกรมต่าง ๆ สามารถติดต่อกับเคอร์นัลได้ ก็จำเป็นต้องอาศัยระบบข่าวสารนั่นเอง

มินิกซ์อนุญาติให้โปรแกรมในระดับของผู้ใช้ปรกติสามารถส่งข่าวสารให้กับส่วนจัดการหน่วยความจำ และระบบไฟล์เท่านั้น นั่นคือ โปรแกรมผู้ใช้จะไม่สามารถส่งข่าวสารไปให้เคอร์เนลโดยตรงได้ ทางแก้ก็คือ โปรแกรมผู้ใช้จะต้องส่งข่าวสารไปให้ส่วนจัดการหน่วยความจำก่อน แล้วให้ส่วนจัดการหน่วยความจำส่งข่าวสารไปให้เคอร์นัลอีกต่อหนึ่ง

เพื่อประโยชน์ในการจัดการเกี่ยวกับชิสเต็มคอลล์ ได้ทำการเพิ่มทาสก์ขึ้นอีกหนึ่งทาสก์ขึ้นภายในเคอร์นัลได้แก่ ไทยทาสก์ ซึ่งซอร์สโค้ดของไทยทาสก์จะอยู่ในไฟล์ thaisys.c การทำงานของไทยทาสก์ จะเหมือนกับทาสก์ทั่ว ๆ ไป คือ จะวนลูปอนันต์รอรับข่าวสารเพื่อที่จะกระโดดไปทำงานที่ต่าง ๆ ตามชนิดของข่าวสารที่ได้รับ ในกรณีของไทยทาสก์นั้น จะเป็นการวนลูปรอรับข่าวสารจากส่วนจัดการหน่วยความจำเท่านั้น

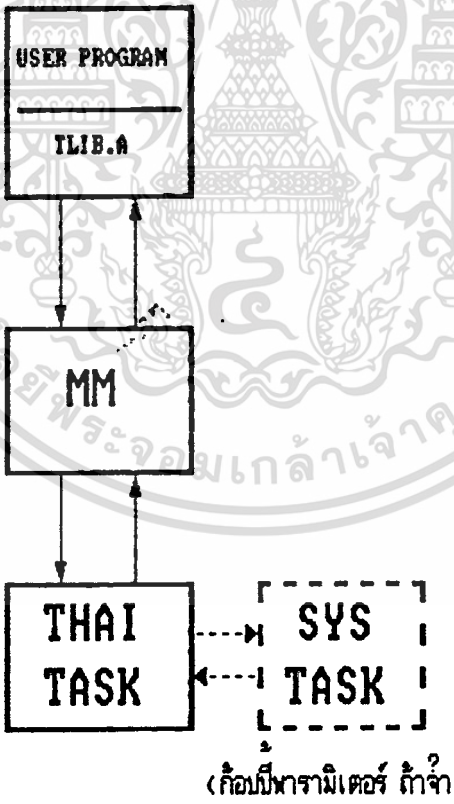
ดังนั้น การทำงานของชิสเต็มคอลล์ จะเป็นลำดับ ๆ คือ

1. โปรแกรมของผู้ใช้ทำการเรียกใช้ฟังก์ชันต่าง ๆ ที่เป็นชิสเต็มคอลล์
2. ฟังก์ชันที่ถูกเรียกใช้ ซึ่งอยู่ใน tlib.a จะทำการส่งข่าวสารไปที่ส่วนจัดการหน่วยความจำ โดยรูปแบบของข่าวสาร จะเป็นไปตามรูปที่ 5.4
3. เมื่อได้รับข่าวสาร ส่วนจัดการหน่วยความจำจะกระโดดไปทำงานที่ฟังก์ชันต่างๆ ตามแต่หมายเลขของชิสเต็มคอลล์ ซึ่งดูได้จากไฟล์ table.c ในส่วน mm และฟังก์ชันการทำงานเกี่ยวกับชิสเต็มคอลล์ที่ได้ทำการเพิ่มเข้าไปใหม่ สามารถดู

ได้จากไฟล์ thai.c การทำงานของฟังก์ชันต่าง ๆ เหล่านี้ก็คือ ทำการส่งข่าวสารต่อไปที่ ไทยทาสก์ ซึ่งอยู่ในส่วนเคอร์นัล โดยรูปแบบของข่าวสารจะเป็นไปตามรูปที่ 5.5

4. เมื่อไทยทาสก์ได้รับข่าวสารก็จะกระโดดไปทำงานตามแต่ซิสเต็มคอลล์แต่ละชนิด และเมื่อทำงานเสร็จ ก็จะส่งข่าวสารกลับไปที่ส่วนจัดการหน่วยความจำ
5. ส่วนจัดการหน่วยความจำ จะส่งข่าวสารกลับไปที่โปรแกรมของผู้ใช้

การทำงานสามารถสรุปได้ดังรูปที่ 5.6



สำหรับการส่งพารามิเตอร์ต่าง ๆ จะส่งไปกับข่าวสาร โดยข่าวสารที่ใช้จะเป็นข่าวสารประเภทที่ 1 (ดูประเภทของข่าวสารทั้ง 6 ประเภทได้ในไฟล์ type.h) ซึ่งมีโครงสร้างเป็นเลขจำนวนเต็ม 3 ตัว และพอยน์เตอร์ชี้ไปที่ข้อมูลตัวอักษร 3 ตัว คือ

m1_i1	m1_i2	m1_i3	m1_p1	m1_p2	m1_p3

ข้อแตกต่างของ ข่าวสารที่ส่งจากโปรแกรมผู้ใช้ไปยังส่วนจัดการหน่วยความจำ กับข่าวสารที่ส่งจากส่วนจัดการหน่วยความจำไปยังไทยทาสก์ ก็คือ ข่าวสารชุดแรก จะไม่มีหมายเลขเทอร์มินัลเข้ามาเกี่ยวข้อง แต่ข่าวสารชุดที่สองโดยมากจะมีหมายเลขเทอร์มินัลอยู่ในฟิลด์ m1_i1 ซึ่งจะใช้ประโยชน์เมื่อมีนักชีมีการต่อเทอร์มินัลหลาย ๆ เครื่อง แต่ในชิสเต็มคอลล์ปัจจุบัน สนับสนุนการใช้งานในแบบเทอร์มินัลเดียวเท่านั้น ๆ ซึ่งจะมีค่าของ T_LINE เป็น 0

สำหรับ ชิสเต็มคอลล์ที่จำนวนพารามิเตอร์ มีมากเกินกว่าที่จะบรรจุไปในข่าวสารได้ ซึ่งได้แก่ Vxyas, Vscrup, Vscrdown และ Vline จำเป็นต้องใช้วิธีพิเศษในการส่งพารามิเตอร์ ซึ่งชิสเต็มคอลล์ทั้ง 4 นี้ ใช้วิธีเดียวกัน ดังนั้นจะอธิบายเป็นตัวอย่างเฉพาะ Vxyas ซึ่งเป็นชิสเต็มคอลล์ที่ซับซ้อนที่สุด

Vxyas เป็นชิสเต็มคอลล์ที่มีรูปแบบการใช้คือ

```
Vxyas(int x,int y,int attr,char *str);
```

มีหน้าที่คือ แสดงผลข้อความ (str) ออกหน้าจอที่ตำแหน่ง (x,y) โดยแสดงผลเป็นสี attr จะเห็นว่าจำนวนพารามิเตอร์ที่เป็นเลขจำนวนเต็มมี 3 ตัว และเมื่อรวมกับหมายเลขเทอร์มินัลอีก 1 ตัว ก็จะได้จำนวนพารามิเตอร์ที่ต้องส่งไปให้ไทยทาสก์ที่เป็นเลขจำนวนเต็มทั้งหมด 4 ตัว ซึ่งจะมากเกินกว่าที่จะใส่ไปในข่าวสารได้ ทางแก้ก็คือ รวมพารามิเตอร์ x,y และ attr เป็นโครงสร้างข้อมูลประเภท xyainfo (อยู่ในไฟล์ type.h) กล่าวคือ

```
struct xyzinfo {
        int x;
        int y;
        int attr;
}
```

เมื่อรวมพารามิเตอร์ทั้งสามเป็นข้อมูลชุดเดียวกันแล้ว ก็จะใช้การส่งแอดเดรสของข้อมูลชุดนี้ไปในข่าวสารแทน และเมื่อข่าวสารไปถึงไทยทาสก์ ไทยทาสก์ก็จะสามารถอ่านค่าของพารามิเตอร์ทั้งสามได้จากแอดเดรสที่ส่งมากับข่าวสาร แต่ปัญหาไม่ได้อยู่แค่นั้น เนื่องจากมินิกซ์เป็นระบบปฏิบัติการแบบมัลติทาสก์ คือสามารถทำงานหลาย ๆ งานพร้อม ๆ กัน ดังนั้นเพื่อที่จะแยกการทำงานของแต่ละโปรเซสออกจากกัน เพื่อให้ทำงานอิสระ ไม่รบกวนกัน จึงต้องทำการอ้างแอดเดรสแบบเสมือน ดังนั้นแอดเดรสที่ส่งมากับข่าวสารก็จะเป็นแอดเดรสแบบเสมือนด้วย ถ้าไทยทาสก์ใช้แอดเดรสนั้นอ่านข้อมูลของพารามิเตอร์ จะได้ข้อมูลที่ไม่ถูกต้อง สาเหตุคือแอดเดรสแบบเสมือนจะอ้างถึงหน่วยความจำของโปรเซสนั้น ๆ เท่านั้น ดังนั้นไทยทาสก์จึงอ่านข้อมูลได้เฉพาะข้อมูลที่อยู่ในไทยทาสก์เท่านั้น แต่พารามิเตอร์ทั้งสามอยู่ในโปรเซสของผู้ใช้ สิ่งที่ไทยทาสก์ทำก็คือ อ่านข้อมูลที่แอดเดรสนั้น ๆ ในหน่วยความจำของไทยทาสก์เอง ซึ่งแน่นอนจะไม่ใช่พารามิเตอร์ที่ต้องการแน่

ทางแก้ก็คือ ต้องทำการก๊อปปี้พารามิเตอร์ทั้งสาม เข้ามาอยู่ในหน่วยความจำส่วนของไทยทาสก์ก่อน และการที่จะทำการก๊อปปี้ได้ ก็จะต้องรู้ว่าพารามิเตอร์ที่ต้องการเป็นของโปรเซสไหน ดังนั้น ส่วนจัดการหน่วยความจำ จึงต้องส่งพารามิเตอร์ who เพิ่มขึ้นมา ให้แก่ไทยทาสก์ เพื่อที่จะบอกไทยทาสก์ว่า โปรเซสใดที่เป็นตัวเรียกใช้ซิสเต็มคอลล์ ซึ่งก็คือ โปรเซสที่ส่งข่าวสารมาที่ส่วนจัดการหน่วยความจำนั่นเอง

เมื่อไทยทาสก์ ทราบทั้งแอดเดรสเสมือน และหมายเลขโปรเซสที่มีพารามิเตอร์อยู่ ก็จะสามารถทำการก๊อปปี้ข้อมูลได้แล้ว โดยแทนที่จะทำการก๊อปปี้เองซึ่งเป็นเรื่องยุ่งยากพอสมควร ไทยทาสก์จะส่งข่าวสาร บอกซิสเต็มทาสก์ให้ทำการก๊อปปี้ข้อมูลให้ ซึ่งเมื่อทำการก๊อปปี้ข้อมูล

เสร็จ ไทยทาสก์ก็จะได้พารามิเตอร์ทั้งสามตามที่ต้องการ

สำหรับข้อความที่ต้องการแสดงผลออกหน้าจอ (str) ก็อยู่ในกรณีเดียวกัน กล่าวคือ จะส่งเฉพาะแอดเดรสของข้อความมากับข่าวสารเท่านั้น ไทยทาสก์ต้องส่งข่าวสารบอกซิสเต็มทาสก์ให้ทำการก๊อปปี้ข้อความมาที่บัฟเฟอร์ในไทยทาสก์ เนื่องจากซิสเต็มทาสก์ จะต้องรู้ขนาดของข้อมูลที่จะทำการก๊อปปี้ ดังนั้นโปรแกรมของผู้ใช้จะต้องส่งความยาวของข้อความมาในข่าวสารด้วย แต่การหาความยาวของข้อความนี้ ไลบรารีใน tlib.a จะจัดการให้เองโดยอัตโนมัติ และจะส่งมากับข่าวสารให้เลย ดังนั้น Vxyas จึงไม่มีพารามิเตอร์บอกความยาวของข้อความ เนื่องจากความจำเป็นในการต้องก๊อปปี้ข้อความมาไว้ที่บัฟเฟอร์ในไทยทาสก์ก่อน ซึ่งเป็นบัฟเฟอร์ที่จำกัดขนาด (MAXBUF) จึงทำให้เกิดข้อจำกัดความยาวของข้อความที่จะแสดงผลโดยใช้ซิสเต็มคอลล์ Vxyas จะต้องยาวไม่เกินขนาดของบัฟเฟอร์

สำหรับซิสเต็มคอลล์ Vxyac มีลักษณะพิเศษที่จำเป็นต้องอธิบาย คือ

Vxyac มีรูปแบบการใช้งานคือ

```
Vxyac(int x,int y,int attr,int ch);
```

มีหน้าที่ คือ แสดงผลตัวอักษร ch ออกหน้าจอที่ตำแหน่ง (x,y) โดยจะแสดงเป็นสี attr จะเห็นว่าซิสเต็มคอลล์นี้มีพารามิเตอร์ 4 ตัวที่เป็นเลขจำนวนเต็ม รวมกับหมายเลขเทอร์มินัล เป็น 5 ตัว จะเห็นว่าไม่พอที่จะใส่ไปในข่าวสาร แต่เนื่องจากในมินิกซ์นี้ ขนาดของข้อมูลแบบจำนวนเต็ม มีขนาดเท่ากับข้อมูลแบบพอยน์เตอร์ชี้ไปยังตัวอักษร ดังนั้นฟิลด์ m1_p1 และ m1_p2 จึงสามารถใช้ใส่ข้อมูลแบบจำนวนเต็มได้ ซึ่งในที่นี้ พารามิเตอร์ attr และ ch จะถูกใส่ไว้ในฟิลด์ทั้งสองนี้ ซึ่งไทยทาสก์ก็จะรู้กัน ว่าในเฉพาะซิสเต็มคอลล์ Vxyac นี้จะใช้ฟิลด์ m1_p1 และ m1_p2 เป็นเสมือนฟิลด์ m1_i4 และ m1_i5 ตามลำดับ

ขั้นตอนในการเพิ่มซิสเต็มคอลล์ใหม่

ถ้าต้องการเพิ่มซิสเต็มคอลล์ใหม่ สามารถทำได้โดยแก้ไข และเพิ่มเติมโค้ดในไฟล์ต่าง ๆ โดยสามารถแบ่งได้เป็น 3 ส่วน คือ

(1) โค้ดของส่วนรวม ซึ่งทั้งส่วนจัดการหน่วยความจำ และเคอร์นัลใช้ร่วมกัน (ไดเร็กตอรี "h")

(2) โค้ดของส่วนจัดการหน่วยความจำ (ไดเร็กตอรี "mm")

(3) โค้ดของเคอร์นัล (ไดเร็กตอรี "kernel")

โดยแต่ละส่วน จะมีขั้นตอนดังนี้ คือ

(1) <u>โค้ดของส่วนรวม</u>

1. เพิ่มโครงสร้างข้อมูลที่จะใช้เข้าไปในไฟล์ h/type.h ถ้ามี
2. เพิ่มชื่อ และหมายเลขของซิสเต็มคอลล์ใหม่ เข้าไปในไฟล์ h/callnr.h
3. เพิ่มชนิดของข่าวสาร ที่ส่วนจัดการหน่วยความจำ จะส่งไปให้ไทยทาสก์ เข้าไปในไฟล์ h/com.h รวมทั้งระบุตำแหน่งของพารามิเตอร์ ในข่าวสารด้วย

(2) <u>โค้ดของส่วนจัดการหน่วยความจำ</u>

1. เพิ่มชื่อฟังก์ชันที่จะจัดการกับซิสเต็มคอลล์ เข้าไปในอาเรย์ mm_callvec ให้ตรงกับหมายเลขของซิสเต็มคอลล์ที่ต้องการเพิ่ม ในไฟล์ mm/table.c
2. ระบุตำแหน่งของพารามิเตอร์ในข่าวสาร ที่โปรเซสของผู้ใช้ส่งมาให้ส่วนจัดการหน่วยความจำ ในไฟล์ mm/param.h
3. เพิ่มโค้ดของฟังก์ชันที่ได้เพิ่มชื่อเข้าไปใน mm_callvec เข้าไปในไฟล์ mm/thai.c โดยฟังก์ชันจะทำหน้าที่คือ เปลี่ยนรูปแบบของข่าวสารที่ได้รับจากโปรเซสของผู้ใช้ แล้วส่งไปให้ไทยทาสก์

(3) <u>โค้ดของส่วนเคอร์นัล</u>

1. เพิ่มฟังก์ชันที่จะจัดการกับข่าวสาร ที่จะได้รับจากส่วนจัดการหน่วยความจำ เข้าไปในไฟล์ kernel/thaisys.c โดยฟังก์ชันนี้จะเป็นส่วนที่ทำงานจริงของ

ซิสเต็มคอลล์

2. อาจเพิ่มฟังก์ชันทำงานต่าง ๆ เข้าไปในไฟล์ kernel/thailib.c ได้ ถ้าต้องการ
3. ทำการแก้ไข เพิ่มเติมส่วนต่าง ๆ ของเคอร์นัล เท่าที่จำเป็นแก่การทำงานของซิสเต็มคอลล์

เมื่อได้ทำการแก้ไข เพิ่มเติมตามขั้นตอนที่กล่าวไปข้างต้นแล้ว ก็ให้ทำการคอมไพล์สร้างระบบใหม่ทั้งส่วนจัดการหน่วยความจำ และส่วนเคอร์นัล แล้วจึงทำการสร้างแผ่นบูตใหม่ ก็จะได้ซิสเต็มคอลล์ใหม่ ตามต้องการ

การเพิ่มไลบรารีเข้าไปในไฟล์ tlib.a

หลังจากที่สร้างแผ่นบูตเรียบร้อยแล้ว ขั้นตอนต่อไปก็คือ ต้องทำการเพิ่มไลบรารีเข้าไปในไฟล์ tlib.a เพื่อให้โปรแกรมของผู้ใช้สามารถเรียกใช้ซิสเต็มคอลล์ได้โดยเปรียบเสมือนกับเป็นชุดคำสั่งที่เพิ่มขึ้นมา ขั้นตอนการทำก็คือ

1. เพิ่มโค้ดฟังก์ชันที่จะให้โปรแกรมของผู้ใช้เรียกใช้ เป็นอีกไฟล์หนึ่งต่างหาก โดยฟังก์ชันนั้น จะทำหน้าที่แปลงพารามิเตอร์ที่ส่งมาจากโปรแกรมผู้ใช้ ให้อยู่ในรูปของข่าวสารที่จะส่งไปให้ส่วนจัดการหน่วยความจำ
2. ทดลองเขียนโปรแกรมทดสอบ เรียกใช้ฟังก์ชันที่ได้เพิ่มขึ้น แล้วทำการคอมไพล์และลิงค์ ดูว่าได้ผลถูกต้องตามที่ต้องการหรือไม่ ถ้าไม่ถูกต้องให้กลับไปทำการแก้ไขให้ถูกต้อง
3. ถ้าถูกต้องแล้ว ให้ทำการเพิ่มฟังก์ชันนั้นเข้าไปในไฟล์ tlib.a โดยคอมไพล์ไฟล์ในข้อ 1 โดยใช้ตัวเลือก -LIB -DTHAI -DI8088 เพื่อให้ได้ไฟล์ประเภท .s ซึ่งเป็นไฟล์แอสเซ็มบลีแบบแพ็ค หลังจากนั้น จึงทำการเพิ่มเข้าไปใน tlib.a โดยใช้โปรแกรม ar โดยมีรูปแบบการใช้คือ

```
$ ar q tlib.a xxx.s
```

หลังจากนั้น ก็สามารถลบไฟล์ .s ที่ได้สร้างขึ้น ทิ้งไปได้ หรือหากมีการแก้ไขโปรแกรมทำให้ได้ .s เวอร์ชันใหม่ ต้องการแก้ไขที่อยู่ใน tlib.a ก็ต้องทำการลบเวอร์ชันเก่าที่อยู่ใน tlib.a ก่อน โดยใช้โปรแกรม ar เช่นเดียวกัน โดยมีรูปแบบการใช้ คือ

```
$ ar d tlib.a xxx.s
```

หลังจากนั้น จึงค่อยทำการเพิ่มเข้าไปในไฟล์ tlib.a ใหม่ โดยใช้วิธีเดิม

m_type	m1_i1	m1_i2	m1_i3	m1_p1	m1_p2
VGETDM					
VSETDM	new_flag				
VDCTRL	new_flag				
VGETKM					
VSETKM	new_mode				
VKCTRL	new_flag				
VKPRESSED					
VSETCOLOR			new_color		
VGETCOLOR					
VXYAS	str_len	who		xyaptr	str_out
VCLRSCR					
VGOTOXY	new_x	new_y			
VGETXY					
VCLRKEY					
VXYAC	new_x	new_y	new_color	char_out	
VSCRDOWN		who		winptr	
VSCRUP		who		winptr	
VDOLINE	linecolor	who		winptr	
VONCURSOR					
VOFFCURSOR					
VSETFONT	res	fontnum	segment	offset	

รูปที่ 5.4 แสดงรูปแบบข่าวสารที่โปรแกรมของผู้ใช้ส่งให้ส่วนจัดการหน่วยความจำ

m_type	m1_i1	m1_i2	m1_i3	m1_p1	m1_p2
VRESETFONT	res	fontnum			
VUSEFONT	res	fontnum			

รูป 5.4 (ต่อ)

m_type	m1_i1	m1_i2	m1_i3	m1_p1	m1_p2
T_GETDM	T_LINE				
T_SETDM	T_LINE	MODE			
T_DCTRL	T_LINE	FLAG			
T_GETKM	T_LINE				
T_SETKM	T_LINE	MODE			
T_KCTRL	T_LINE	FLAG			
T_KPRESSED	T_LINE				
T_SETCOLOR	T_LINE		COLOR		
T_GETCOLOR	T_LINE				
T_XYAC	T_LINE	POSX	POSY	COLOR	CHAR
T_XYAS	T_LINE	STRLEN	WHO	XYAPTR	STRING
T_CLRSCR	T_LINE				
T_GOTOXY	T_LINE	POSX	POSY		
T_GETXY	T_LINE	AXIS			
T_CLRKEY	T_LINE				
T_SCRUP	T_LINE		WHO	WINPTR	
T_SCRDOWN	T_LINE		WHO	WINPTR	
T_DOLINE	T_LINE	LINECOL-	WHO	WINPTR	
T_ONCURS-	T_LINE				
T_OFFCUR-	T_LINE				

รูปที่ 5.5 แสดงรูปแบบข่าวสารที่ส่วนจัดการหน่วยความจำส่งให้ไทยทาสก์

m_type	m1_i1	m1_i2	m1_i3	m1_p1	m1_p2
T_SETFONT	T_LINE	RES	FONTNUM	SEGMENT	OFFSET
T_RESETFONT	T_LINE	RES	FONTNUM		
T_USEFONT	T_LINE	RES	FONTNUM		

รูปที่ 5.5 (ต่อ)

การแก้ไขส่วนคอมมานด์เชลล์ให้รับข้อความภาษาไทย

หลังจากทดลองสร้างระบบปฏิบัติการมินิกซ์ภาษาไทยได้แล้ว ได้ทำการทดสอบการทำงานในส่วนที่เกี่ยวข้องกับการอ่าน/เขียนข้อความภาษาไทย ปรากฏว่าสามารถป้อนข้อความภาษาไทยได้ ทั้งในโหมด 17 และ 25 บรรทัด รวมทั้งสามารถ cat ไฟล์ที่เก็บข้อความภาษาไทยได้ แต่เมื่อมีการผ่านข้อความภาษาไทยผ่านเชลล์ ปรากฏว่าเชลล์จะทำการมาสก์บิต 7 ของตัวอักษรภาษาไทยทุกตัว ทำให้ข้อความเหล่านั้นกลายเป็นขยะ เช่น

```
# echo "กขคง"
!@#$ : not found
```

ดังนั้นจึงได้มีการแก้ไขซอร์สโค้ดเชลล์ โดยตัดส่วนที่มาสก์บิต 7 ออก และถือว่าตัวอักษรภาษาไทยเป็นตัวอักษร alphabetic ด้วย โดยแก้ไขที่ไฟล์ sh.h, sh1.c, sh3.c โดยใช้ข้อกำหนดในการแก้ไขเหมือนกับส่วนอื่นๆ คือ การเพิ่มโค้ดโดยตรวจดูการนิยามสัญญลักษณ์ THAI ดังนั้น ถ้าต้องการคอมไพล์เชลล์เดิม ก็สามารถทำได้โดยการตัดออปชัน -D THAI ในการคอมไพล์โปรแกรมเชลล์ออก

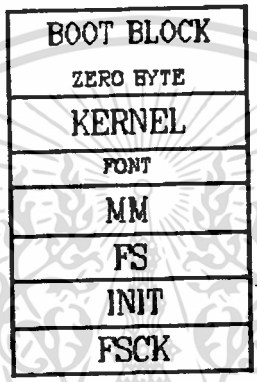
หลังจากแก้ไขเชลล์แล้ว ปรากฏว่าเชลล์สามารถรับข้อความภาษาไทยได้ โดยสามารถสร้างไฟล์ที่มีชื่อไฟล์เป็นภาษาไทยได้ สามารถสร้างยูสเซอร์ชื่อภาษาไทยได้ และสามารถป้อนข้อความภาษาไทยไปเก็บลงยังไฟล์ได้ เช่น

```
# echo "เบิร์ด พริกขี้หนู" > ธงชัย
# cat ธงชัย
เบิร์ด พริกขี้หนู
```

บทที่ 6
การสร้างแผ่นบู๊ตดิสค์และแผ่นรูตไฟล์ซิสเต็ม

การสร้างแผ่นดิสต์สำหรับบุคมินิกซ์

แผ่นดิสต์สำหรับบุดแบ่งออกเป็น 6 ส่วน ดังนี้



รูป 6.1 แสดงโครงสร้างของแผ่นบู๊ตดิสค์

การสร้างแผ่นจะใช้โปรแกรม build ซึ่งอยู่ใน /tools โดยทั้ง 6 ส่วน ได้มาจากการคอมไพล์ source code ทั้ง 6 ส่วนเป็นไฟล์ต่างๆ โดยนำมาไว้รวมกันใน /tools ด้วย แล้วทำตามขั้นตอนดังนี้

1. ทำการ umount ไดรส์ A: (/dev/fd0)

```
/etc/umount /dev/fd0
```

2. จากนั้น mount /tools เข้าไดรส์ B: ที่ /usr

```
/etc/mount /dev/fd1 /usr
```

3. เข้าไปอยู่ใน /tools

```
cd /usr/tools
```

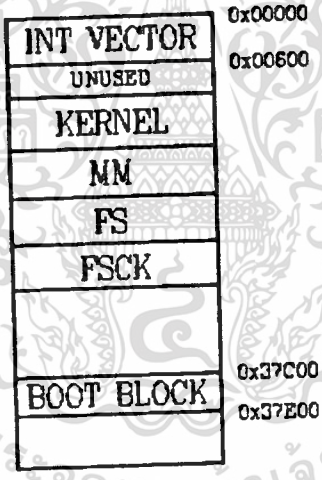
4. นำแผ่นเปล่าที่ฟอร์แมตเรียบร้อยแล้วใส่ในไดรส์ A: แล้วนิมพ์คำสั่งดังนี้

```
build booblk kernel mm fs init fsck /dev/fd0
```

การทำงานของโปรแกรม build

โปรแกรม build จะนำส่วนต่างๆมาเรียงต่อกัน โดยส่วน boot block จะต้องมีขนาด 512 ไบท์เสมอ แต่ถ้าตัวโปรแกรมมีขนาดไม่ถึง ส่วนที่เหลือใส่เป็นศูนย์แทน สำหรับส่วน kernel mm fs initt fsck ขนาดจะมีหน่วยเป็นพารากราฟ (16 ไบต์ = 1 พารากราฟ) ซึ่งทั้ง 5 ส่วนนี้แต่ละส่วนจะประกอบด้วย text กับ data แยกกัน

ในขณะที่บูตระบบมินิกซ์ คอมพิวเตอร์จะโหลดส่วน boot block ที่อยู่ในแผ่นบู๊ตเข้าไปในหน่วยความจำแอดเดรส 0x37C00 เพื่อโหลดส่วนอื่นๆตามมา จากนั้นส่วน boot block จะโหลดส่วนเคอร์เนลเข้าไปในหน่วยความจำเริ่มที่แอดเดรส 0x600 (1536) จากนั้นก็จะโหลดส่วน mm fs fsck เรียงตามกันมา ซึ่งในขณะนี้หน่วยความจำจะมีลักษณะดังรูป 6.2



รูป 6.2 โครงสร้างของหน่วยความจำเมื่อเริ่มเข้าสู่มินิกซ์

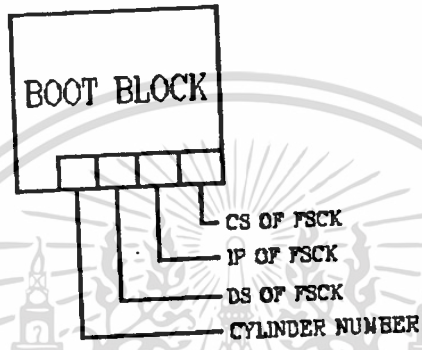
แล้วส่วน boot block จะส่งการทำงานให้แก่ส่วน fsck ซึ่งจะแสดงเมนูต่างๆบนหน้าจอให้ผู้ใช้เลือกการทำงาน ถ้าผู้ใช้กดคีย์ "=" fsck จะส่งการทำงานให้ส่วนเคอร์นัล เพื่อเข้าสู่ระบบมินิกซ์

จะเห็นได้ว่าการที่ส่วน boot block จะทำการโหลดส่วนต่างๆเข้าสู่หน่วยความจำแล้วส่งการทำงานให้แก่ส่วน fsck ได้นั้น จะต้องรู้ข้อมูลดังนี้

1. จำนวน cylinder ที่จะโหลดส่วนต่างๆ เข้ามาในหน่วยความจำ

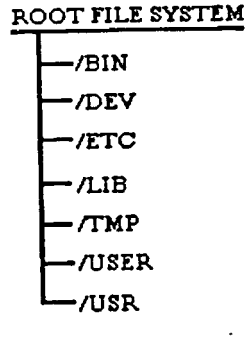
2. ตำแหน่งแอดเดรสเริ่มต้นของส่วน fsck ที่จะเริ่มทำงาน

ข้อมูลทั้งสองนี้ เป็นหน้าที่ของโปรแกรม build ซึ่งในขณะที่กำลังสร้างแผ่นบู๊ต โปรแกรม build จะต้องอ่านไฟล์ส่วนต่างๆ แล้วคำนวณจำนวน cylinder จากนั้นจะเก็บไว้ที่ 4 words สุดท้ายของส่วน boot block ดังรูป 6.3



รูป 6.3 แสดงตำแหน่งเริ่มต้นที่เก็บข้อมูลของส่วน fsck

ในกรณีผู้ใช้กดคีย์ "=" เพื่อเข้าสู่ระบบมีนิกซ์ fsck จะส่งการทำงานให้ส่วน kernel ซึ่งเป็นการเริ่มต้นการทำงานที่แอดเดรส 0x600 (1536) ก่อนที่จะกดคีย์ "=" ผู้ใช้จะต้องเปลี่ยนแผ่นบู๊ตเป็นแผ่น root file system จากนั้น ส่วน kernel จะโหลดส่วน init เข้าไปในหน่วยความจำ โดยทับส่วน fsck แล้วโหลด ram disk ต่อท้ายซึ่ง ram disk จะประกอบด้วย root file system ดังรูป 6.4



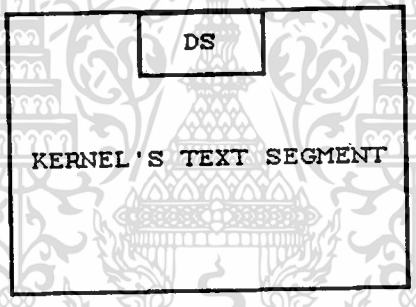
รูป 6.4 แสดงรูตไฟล์ซิสเต็ม

จากนั้น kernel จะทำงานตามโปรแกรม system initialization ซึ่งอยู่ใน /etc/rc โดยจะสั่งผู้ใช้เปลี่ยนแผ่น root file system มาเป็นแผ่น user file system เพื่อทำการ mount ไปไว้ใน /usr นั่นคือ ส่วนเคอร์นัลจะทำคำสั่ง "/etc/mount /dev/fd0 /usr" เมื่อ mount เสร็จ โปรแกรมจะให้ใส่วันที่ แล้วเรียกโปรแกรม login ซึ่งก็เป็นการเข้าสู่ระบบมินิกซ์เป็นที่เรียบร้อย

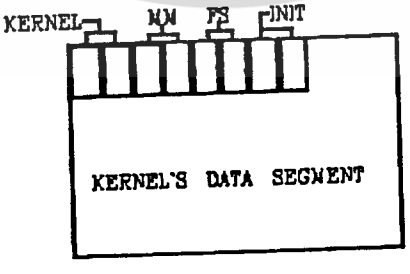
การที่ส่วน kernel จะโหลดส่วน init และ ram disk ได้นั้น ต้องรู้ข้อมูลดังนี้

1. ตำแหน่งแอดเดรสเริ่มต้นของ fsck ซึ่งจะโหลดส่วน init

2. ตำแหน่งแอดเดรสที่จะโหลดส่วน ram disk ต่อท้ายส่วน init ซึ่งข้อมูลเหล่านี้ โปรแกรม build จะเป็นตัวจัดการ ดังรูป 6.5

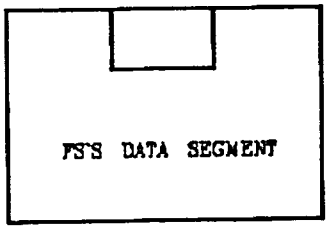


ตำแหน่งตาต้าเซกเมนต์ของเคอร์นัล



รูปแสดงโค้ดเซกเมนต์และดาต้าเซกเมนต์ของส่วนเคอร์นัล, MM, FS, INIT

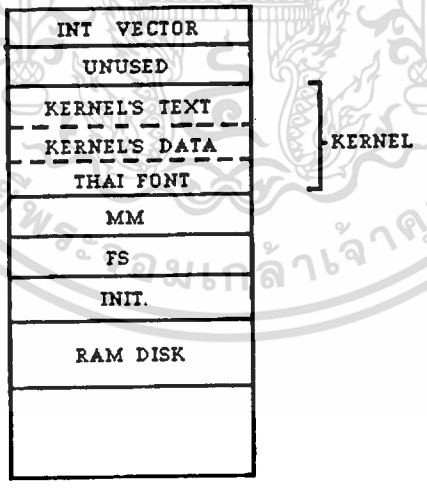
รูป 6.5 ข้อมูลในส่วนต่างๆที่ใช้ในระหว่างการบู๊ตมินิกซ์



ตำแหน่งเริ่มต้นและขนาดของส่วน INIT

รูป 6.5 ต่อ

สำหรับการฝังภาษาไทยลงในระบบมินิกซ์ ซึ่งประกอบด้วยฟอนต์ภาษาไทย/อังกฤษ เข้าเป็นข้อมูลส่วนหนึ่งของส่วน kernel ซึ่งโครงสร้างของหน่วยความจำจะเป็นดังรูป 6.6



รูป 6.6 แสดงโครงสร้างของหน่วยความจำหลังจากเข้าสู่ระบบมินิกซ์แล้ว

การสร้างฟอนต์ไฟล์

ไฟล์ font ที่ใช้ในการสร้างแผ่นบู๊ตดิสค์ จะประกอบไปด้วยฟอนต์ขนาด 256 ตัวอักษรทั้งสิ้น 4 ชุด โดย 2 ชุดเป็นฟอนต์ขนาด 8x20 ตัวอักษร และอีก 2 ชุดเป็นฟอนต์ขนาด 8x14 ตัวอักษร การออกแบบฟอนต์สามารถทำได้โดยใช้โปรแกรมออกแบบฟอนต์ทั่วๆไปโดยจัดเก็บฟอนต์เป็นแบบข้อมูลไบนารีธรรมดาที่ไม่มีเฮดเดอร์ ซึ่งขนาดของฟอนต์แต่ละขนาด และขนาดฟอนต์รวมได้กล่าวไว้แล้วในบทที่ 4 โดยการสร้างไฟล์ font จากการรวมฟอนต์ทั้งสี่ จะกระทำได้ตามลำดับดังนี้

```
# cat font8x20_1 >font
# cat font8x20_2 >>font
# cat font8x14_1 >>font
# cat font8x14_2 >>font
```

เอาท์พุทไฟล์ font จะประกอบไปด้วยฟอนต์ขนาด 8x20 ตัวอักษรทั้งหมด 2 ชุด แล้วตามด้วยฟอนต์ขนาด 8x14 ตัวอักษรอีก 2 ชุด และจะมีขนาดรวมทั้งสิ้น 17,408 ไบต์

การสร้างแผ่นรู๊ตไฟล์ซิสเต็ม

แผ่นรู๊ตไฟล์ซิสเต็มจะเป็นแผ่นที่สองที่ใช้ในการบู๊ตระบบปฏิบัติการมินิกซ์ โดยเป็นแผ่นที่เก็บอิมเมจ (image) ของระบบไฟล์นับจากราก (root file system) มินิกซ์จะจองเนื้อที่แรมดิสค์เท่ากันกับขนาดของระบบไฟล์ในแผ่นนี้ และจะก็อปปี้ข้อมูลทั้งหมดจากแผ่นดิสค์เข้าสู่แรมดิสค์ ทั้งไฟล์และไดเรกทอรีทั้งหมดที่อยู่ในแผ่นนี้จะถูกก็อปปี้ไปไว้ในแรมดิสค์เริ่มจากราก (/) เป็นต้นไป ดังนั้นการสร้างแผ่นรู๊ตไฟล์ซิสเต็มจึงมีความสำคัญมากในการใช้งานมินิกซ์

การสร้างแผ่นรู๊ตไฟล์ซิสเต็ม ที่ใช้ในการพัฒนาจะแบ่งเป็น 2 รุ่น ด้วยกัน คือรุ่นที่ใช้กับเครื่องเอ็กซ์ที และรุ่นที่ใช้กับเครื่องเอทีที่มีไดรฟ์ความจุ 1.2M อย่างน้อยหนึ่งตัว วิธีการสร้างแผ่นรู๊ตไฟล์ซิสเต็มเริ่มแรกจะต้องหาแผ่นดิสค์เปล่าเพื่อที่จะสร้างระบบไฟล์ขึ้นมาก่อนด้วยโปรแกรม mkfs หรือสร้างจากเมนูเมื่อบู๊ตมินิกซ์ก็ได้ สำหรับการเรียกใช้ mkfs จะเป็นดังนี้

1) สำหรับเครื่องเอ็กซ์ทีและมีไดรฟ์ความจุ 360K

```
mkfs /dev/fd0 200
```

เป็นการสร้างระบบไฟล์ขนาด 200K ลงในแผ่น สำหรับขนาดของระบบไฟล์จะมีขนาดเท่าใดก็ได้ แต่จะต้องไม่เกิน 256K ขอให้เลือกให้เหมาะสมเพื่อความสะดวกในการใช้งานโปรแกรม เพราะถ้าเลือกขนาดระบบไฟล์ใหญ่มาก ก็จะทำให้เหลือเนื้อที่หน่วยความจำสำหรับโปรเซสต่างๆน้อยลง และเสียเวลาในการ initialize มาก แต่ถ้าเลือกขนาดระบบไฟล์เล็กเกินไป ก็จะเก็บไฟล์ที่ใช้บ่อยๆได้ไม่มาก ระบบจะต้องอ่านโปรแกรมจากฟลอปปี้ดิสค์มากขึ้นทำให้ระบบทำงานช้าลง

2) สำหรับเครื่องเอที ไดรฟ์ความจุ 1.2M และมีหน่วยความจำเกิน 1 เมกกะไบต์

```
mkfs /dev/at0 544
```

เป็นการสร้างระบบไฟล์ขนาด 544K ลงในแผ่น สำหรับขนาดแปรได้ไม่เกิน 1.2M โดยถ้าขนาดระบบไฟล์มีขนาดเกิน 256K ระบบจะจองหน่วยความจำขยาย (extended memory) สำหรับเก็บระบบไฟล์รากโดยอัตโนมัติ แต่ถ้าไม่มีหน่วยความจำขยายจะต้องเซตขนาดระบบไฟล์ได้ไม่เกิน 256K

หลังจากได้ระบบไฟล์ว่างๆแล้ว ให้เมาน์ทแผ่นนี้เข้าสู่ระบบไฟล์ แล้วทำการสร้างไฟล์และไดเรกทอรีดังต่อไปนี้ ไว้ในระบบไฟล์ด้วย

1) ไฟล์ .ellepro.bl, .profile

2) ไดเรกทอรี bin, dev, etc, lib, tmp, user, usr

โดยไฟล์ .ellepro.bl จะมีหรือไม่มีก็ได้ ดูรายละเอียดได้ในภาคผนวก ช. การใช้อิดิตเตอร์แอลเล่ และไฟล์ .profile จะมีหรือไม่มีก็ได้เช่นกัน ดูรายละเอียดได้ในภาคผนวก ง. การพัฒนาโปรแกรมบนมินิกซ์

ในแต่ละไดเรกทอรีควรเก็บไฟล์ที่เหมาะสมไว้ตามความจำเป็นและขนาดของระบบไฟล์ โดยทำการก็อปปี้จากระบบไฟล์รากในขณะนั้น แต่สำหรับไดเรกทอรี dev ไม่สามารถใช้การก็อปปี้โดยตรงได้ เพราะแต่ละไฟล์ในไดเรกทอรีนี้เป็นไฟล์พิเศษ (ไฟล์ดีไวซ์) จะต้องสร้างโดยการเรียกโปรแกรม mknod โดยกำหนดชื่อดีไวซ์ ชนิดของดีไวซ์ หมายเลขดีไวซ์ทั้งเมเจอร์และหมายเลขไมเนอร์ ดังนี้

```
# mknod /dev/at0 b 2 0
# mknod /dev/at1 b 2 1
# mknod /dev/console c 4 0
# mknod /dev/fd0 b 2 0
# mknod /dev/fd1 b 2 1
# mknod /dev/kmem c 1 2
# mknod /dev/ip c 6 0
# mknod /dev/mem c 1 1
# mknod /dev/null c 1 3
# mknod /dev/ram b 1 0
# mknod /dev/tty c 5 0
# mknod /dev/tty0 c 4 0
# mknod /dev/tty1 c 4 1
```

สำหรับพาธ /lib จะเก็บคอมไพเลอร์ภาษาซีส่วนหนึ่ง รวมทั้งไลบรารีไว้ ถ้ายังมีเนื้อที่ระบบไฟล์เหลือเพียงพอ (กรณีระบบไฟล์ที่มีขนาด 544K ขึ้นไป) และปล่อยว่างไว้สำหรับกรณีอื่นๆ และพาธ /user และ /usr จะปล่อยว่างๆไว้เพื่อเป็นจุดที่ใช้สำหรับเมานท์แผ่นจาก

ไดรฟ์ A และไดรฟ์ B

สุดท้าย ควรจะเหลือเนื้อที่ของแผ่นรู๊ตไฟล์ซิสเต็มไว้ประมาณ 50K เพื่อใช้ในการเก็บไฟล์ชั่วคราวสำหรับคอมไพเลอร์ หลังจากนั้นก็เป็นอันเสร็จกระบวนการสร้างแผ่นรู๊ตไฟล์

เมื่อสร้างแผ่นรู๊ตไฟล์ซิสเต็มได้แล้ว ให้ลองทดสอบการใช้งานดู โดยการบู๊ตระบบใหม่และใช้แผ่นนี้เมื่อระบบให้ใส่แผ่นรู๊ตไฟล์ซิสเต็ม เมื่อระบบโหลดแรมดิสค์เสร็จแล้ว ให้ลองใช้งานดูเพื่อหาค่าขนาดของระบบไฟล์ให้เหมาะสมกับการใช้งาน แล้วสร้างแผ่นรู๊ตไฟล์ซิสเต็มใหม่จนกว่าจะเหมาะสมกับอุปกรณ์และฮาร์ดแวร์ที่มีอยู่

บทที่ 7
โปรแกรมประยุกต์ใช้งานเชลล์

โปรแกรมเชลล์ เป็นตัวอย่างโปรแกรมประยุกต์ใช้งานบนระบบปฏิบัติการมินิกซ์ภาษาไทย และเป็นคนละโปรแกรมกับคอมมานด์เชลล์ (sh) โดยมีความสามารถในการช่วยอำนวยความสะดวกในการเรียกใช้คอมมานด์เชลล์อื่นๆ เช่น ls, cp, rm ผ่านทางระบบเมนูที่มีความสามารถสูง (ใช้พูลดาวน์เมนู, ป็อปอัพวินโดว์, dialog box เป็นต้น) และมีคำอธิบายช่วยเหลือ การใช้คำสั่งต่างๆเหล่านี้ โดยไม่จำเป็นจะต้องไปเปิดคู่มือการใช้คำสั่งคอมมานด์เอง

รายละเอียดของ SHELL

หลังจากที่นำระบบภาษาไทย ฝังเข้าไปเป็นส่วนหนึ่งของระบบปฏิบัติการ (minix) เป็นที่เรียบร้อยแล้ว เพื่อทำการทดสอบการใช้งานระบบภาษาไทย จึงได้เขียนโปรแกรม shell ขึ้นมา ซึ่งมีคุณสมบัติดังนี้

1. ประกอบด้วยกลุ่มคำสั่งต่างๆ จำนวน 5 กลุ่ม คือ

- กลุ่มคำสั่งจัดการกับระบบแฟ้มข้อมูล (file management) ได้แก่

คำสั่ง copy	:	คัดลอกแฟ้มข้อมูล
คำสั่ง ren	:	เปลี่ยนชื่อแฟ้มข้อมูล
คำสั่ง del	:	ลบแฟ้มข้อมูลออกจากระบบ
คำสั่ง cmp	:	เปรียบเทียบข้อมูลในแฟ้มข้อมูล 2 แฟ้ม
คำสั่ง change	:	เปลี่ยนโหมดของแฟ้มข้อมูล
คำสั่ง grep	:	ค้นหาข้อมูลที่ต้องการในแฟ้มข้อมูล

- กลุ่มคำสั่งจัดการกับระบบไดเรคทอรี่ (directory management) ได้แก่

คำสั่ง dir : แสดงรายชื่อแฟ้มข้อมูลต่างๆที่เก็บในไดเรกทอรี่

คำสั่ง makd : สร้างไดเรกทอรี่ขึ้นใหม่

คำสั่ง rend : ลบไดเรกทอรี่ออกจากระบบ

คำสั่ง mount : เพิ่มระบบแฟ้มข้อมูลเข้าในแรมดิสก์

คำสั่ง umount : ลบระบบแฟ้มข้อมูลออกจากแรมดิสก์

- กลุ่มคำสั่งจัดการกับระบบดิสก์ (disk management)

คำสั่ง free : แสดงจำนวนบล็อคที่เหลือในแผ่นดิสก์

คำสั่ง makef : สร้างระบบแฟ้มข้อมูลลงในแผ่นดิสก์

- กลุ่มคำสั่งจัดการกับระบบการพิมพ์ (print management)

คำสั่ง print : พิมพ์แฟ้มข้อมูลออกทางเครื่องพิมพ์

- กลุ่มคำสั่งจัดการทั่วๆไป (general management)

คำสั่ง date : แสดงและแก้ไขข้อมูลเกี่ยวกับวันที่

2. การเลือกใช้คำสั่งใดๆ ใช้ระบบเมนู (menu system) ซึ่งมีคุณสมบัติดังนี้

- เป็นระบบเมนูชนิด pull down ประกอบด้วย กลุ่มคำสั่งหลัก 5 กลุ่ม การเลือกใช้งาน ให้ใช้คีย์ลูกศรขวา,ซ้าย หรือใช้ระบบ hotkey ก็ได้ แต่โดยทั่วไป hotkey ที่ใช้คือ คีย์ alt ร่วมกับ คีย์อักษรตัวใหญ่ของชื่อกลุ่มคำสั่งหลัก เช่น alt-f ใช้เลือกกลุ่มคำสั่งที่จัดการเกี่ยวกับระบบแฟ้มข้อมูล สำหรับแต่ละกลุ่มคำสั่งหลักจะประกอบด้วยคำสั่งย่อยๆอีก เลือกใช้โดยใช้คีย์ลูกศรขึ้น,ลง หรือ คีย์ตัวอักษรตัวแรกของชื่อคำสั่งย่อยนั้นๆ

- ประกอบด้วยระบบ message แสดงอยู่กึ่งกลางบรรทัดสุดท้ายของจอภาพตลอดเวลา ในขณะกำลังเลือกใช้งานคำสั่งต่างๆ

- ประกอบด้วยระบบ hotkey คือการใช้คีย์บอร์ดพิเศษ หรือ คีย์บอร์ดรวม (combined key) ในการเรียกใช้งานคำสั่งที่ต้องการอย่างรวดเร็ว โดยไม่ต้องใช้คีย์ลูกศร เช่น คีย์ ctrl-c เป็น hotkey ในการเรียกใช้คำสั่ง copy แฟ้มข้อมูล เป็นต้น

3. ในการเรียกใช้คำสั่งย่อยต่างๆ จะต้องมีการให้ค่าพารามิเตอร์แก่คำสั่งนั้นๆด้วย จึงได้มีการสร้างระบบรับข้อมูล (พารามิเตอร์) เรียกกันโดยทั่วไปว่า ระบบ dialog box ซึ่งมีคุณสมบัติดังนี้

- มีลักษณะเป็น popup window

- แบ่งเป็น 2 ส่วนหลักคือ ส่วนรับข้อมูล (edit) กับ ส่วนเลือกข้อมูล (select) โดยส่วนแรก จะเรียงอยู่ตอนต้นของ dialog box แล้วจึงตามด้วยส่วนเลือกข้อมูล

- สำหรับการเลือกใส่ข้อมูล ใช้คีย์ tab เป็นคีย์เลือก และ ในส่วนเลือกข้อมูล ให้ใช้คีย์ enter เลือกว่าต้องการหัวข้อนี้ ถ้าไม่ต้องการให้กดคีย์ enter ซ้ำ ซึ่งมีการทำงานเป็นลักษณะ toggle นั่นเอง

4. shell เป็นโปรแกรมใช้งาน ประกอบด้วยคำสั่งต่างๆมากมาย ซึ่งมีอยู่ในมินิกซ์อยู่แล้ว ดังนั้นเพื่อมิให้เป็นการทำงานซ้ำซ้อน จึงเรียกใช้งานคำสั่งต่างๆ โดยใช้ system call ที่ชื่อว่า exec แล้วทำการเปลี่ยนทิศทางของผลลัพธ์ที่เกิดจากการเรียกใช้งานคำสั่งนั้นๆ ซึ่งปกติแสดงออกทางจอภาพ ให้แสดงลงในแฟ้มข้อมูลแทน จากนั้นก็เป็นหน้าที่ของโปรแกรม shell ที่จะต้องแสดงผลลัพธ์นั้นออกทางจอภาพเอง

5. จากการที่โปรแกรม shell ต้องทำการแสดงผลลัพธ์ที่ได้จากการไปเรียกใช้งานคำสั่ง

ซึ่งมีอยู่แล้วในมีนิกซ์ ทำให้ต้องมีส่วนแสดงผลข้อมูลภายในโปรแกรม shell ด้วย จึงได้สร้างระบบ browser ขึ้นมาสำหรับแสดงผลข้อมูล โดยมีคุณสมบัติดังนี้

- มีลักษณะเป็น popup window
- สามารถใช้คีย์ลูกศรขึ้น,ลง,ซ้าย,ขวา เลื่อนไปดูส่วนใดๆของ window ก็ได้
- สามารถดูข้อมูลทีละหน้า (page) โดยใช้คีย์ page-up, page-down ได้
- สามารถเลื่อนดูข้อมูลแบบต่อเนื่อง ทั้งก่อนและหลังข้อมูลของ window ในขณะนั้นได้ (scroll up และ scroll down) โดยใช้คีย์ลูกศรขึ้น,ลง
- สามารถเลื่อนดูข้อมูลแบบต่อเนื่อง ทั้งในแนวขวาและซ้ายข้อมูลของ window ในขณะนั้นได้ (scroll right และ scroll left) โดยใช้คีย์ลูกศรขวา,ซ้าย
- สามารถดูข้อมูลตอนต้นได้ทันที โดยไม่ต้องกดคีย์ page-up หลายครั้ง โดยใช้คีย์ home (top of data)
- สามารถดูข้อมูลสุดท้ายได้ทันที โดยไม่ต้องกดคีย์ page-down หลายครั้ง โดยใช้คีย์ end (end of data)

โครงสร้างข้อมูลของ ระบบ SHELL

จากที่ได้อธิบายถึงลักษณะ shell แล้ว สามารถจัดโครงสร้างข้อมูลของระบบ shellได้โดยแบ่งออกเป็น 2 ส่วน ดังนี้

1. โครงสร้างข้อมูล (data structure) ของส่วนระบบเมนู (menu system)
2. โครงสร้างข้อมูล (data structure) ของส่วนระบบรับข้อมูล (dialog box)

(1) โครงสร้างข้อมูลของส่วนระบบเมนู (menu system)

ประกอบด้วยโครงสร้างย่อยดังนี้

- โครงสร้างของระบบเมนูชนิด pull down
- โครงสร้างของระบบ hotkey

โครงสร้างข้อมูลของระบบเมนูชนิด pull down

ระบบเมนูชนิด pull down ประกอบด้วย 2 ส่วนหลัก คือ

1.1 ส่วนกลุ่มการทำงานหลัก (main menu) ซึ่งจะมีจำนวนกลุ่มเท่าไรนั้นขึ้นอยู่กับการนำระบบเมนูชนิดนี้ไปใช้งาน ซึ่งในที่นี้ได้นำระบบเมนูไปใช้ในโปรแกรม shell ทำให้แบ่งกลุ่มการทำงานหลัก หรือ กลุ่มคำสั่งต่างๆนั่นเอง ออกเป็น 5 กลุ่ม ดังที่ได้กล่าวมาแล้ว

1.2 ส่วนกลุ่มการทำงานย่อย (sub menu) กลุ่มการทำงานหลักแต่ละกลุ่มต้องประกอบด้วยกลุ่มการทำงานย่อย ซึ่งกลุ่มการทำงานย่อยแต่ละกลุ่ม ก็ยังสามารถประกอบด้วยกลุ่มการทำงานย่อยๆลงไปได้อีกไม่สิ้นสุด ซึ่งก็เป็นไปตามลักษณะของระบบเมนูชนิด pull down ดังนั้นในโครงสร้างของส่วนกลุ่มการทำงานย่อยนี้ จึงมีลักษณะเป็น recursive structure

โครงสร้างของระบบเมนู pulldown จึงแบ่งออกเป็น 2 โครงสร้าง ดังนี้

1.1 โครงสร้างของส่วนกลุ่มการทำงานหลัก (main menu)

```
struct bar_
  {
    unsigned char **name ; /* ตัวแปรเก็บภาพของส่วนกลุ่มการทำงานหลัก */
    unsigned char **help ; /* ตัวแปรเก็บระบบ message ของหัวข้อการทำงานหลัก
                              แต่ละหัวข้อ */
```

```
    unsigned char    *key ; /* ตัวแปรเก็บระบบ  hotkey ของหัวข้อการทำงานหลัก
                               แต่ละหัวข้อ */
    struct menublk **sub ; /* ตัวแปรเก็บกลุ่มการทำงานย่อย ของแต่ละหัวข้อการทำ
                               งานหลัก */
} ;
```

1.2 โครงสร้างของส่วนกลุ่มการทำงานย่อย (sub menu)

```
struct menublk
 {
    unsigned char     **name ; /* ตัวแปรเก็บภาพของส่วนการทำงานย่อย */
    unsigned char     **help ; /* ตัวแปรเก็บระบบ  message ของกลุ่มการทำงาน
                                  ย่อยแต่ละกลุ่ม */
    unsigned int       *buf ; /* ตัวแปรเก็บรายละเอียดของภาพเดิมบนจอภาพ ที่
                                  เมนูไปวาดทับ */
             int (**func)() ; /* ตัวแปรเก็บฟังก์ชั่นการทำงานของแต่ละคำสั่งย่อย
                                 */
             int          wd ; /* ขนาดความยาวของเมนู มีค่าตั้งแต่ 0 ถึง 89 */
             int          ht ; /* ขนาดความกว้างของเมนู มีค่าตั้งแต่ 0 - 24 */
             int         row ; /* ตำแหน่งแถวมุมบนซ้ายมือของเมนู ที่จะวาดบนจอ
                                  ภาพ มีค่าตั้งแต่ 0 ถึง 24 */
             int         col ; /* ตำแหน่งคอลัมน์มุมบนซ้ายมือของเมนู ที่จะวาดบน
                                  จอภาพ มีค่าตั้งแต่ 0 ถึง 89 */
             int    lastvsel ; /* ที่จำคำสั่งย่อยในเมนู ซึ่งถูกใช้เป็นครั้งสุดท้าย */
    struct menublk    **next ; /* ตัวแปรเก็บกลุ่มการทำงานย่อยของ แต่ละคำสั่งใน
                                  กลุ่มการทำงานย่อย  ซึ่งทำให้เกิดเป็นระบบเมนู
```

```
                                ชนิด  pull down */
} ;
```

โครงสร้างข้อมูลของระบบ hotkey

ใช้หลักการที่ว่า hotkey หนึ่งๆ ใช้แทนคีย์ย่อยๆ ที่ต้องกดเป็นจำนวนหลายคีย์ ซึ่งมีโครงสร้างข้อมูลดังนี้

```
struct hotkey_
  {
    int     hot ; /* รหัส hotkey */
    int     num ; /* จำนวนคีย์ย่อย ที่ถูกแทนด้วยการกด hotkey */
    int sim[10] ; /* ตัวแปรที่เก็บ รหัสคีย์ย่อยๆ ที่ถูกแทนด้วย hotkey */
  } ;
```

(2) โครงสร้างข้อมูลของส่วนระบบรับข้อมูล (dialog box)

ประกอบด้วยโครงสร้างย่อยดังนี้

- โครงสร้างของส่วนรับข้อมูล (edit)
- โครงสร้างของส่วนระบบรับข้อมูล (dialog box)

โครงสร้างของส่วนรับข้อมูล (edit)

เนื่องจากส่วนระบบรับข้อมูล (dialog box) แยกเป็น 2 ส่วน คือ ส่วนรับข้อมูล (edit) กับ ส่วนเลือกข้อมูล (select) ดังนั้น จึงต้องมีโครงสร้างข้อมูลของทั้งสองส่วน แต่โครงสร้างข้อมูลของส่วนเลือกข้อมูล (select) อยู่รวมกับส่วนระบบรับข้อมูล (dialog box)

แล้ว โครงสร้างของส่วนรับข้อมูล (edit) มีลักษณะดังนี้

```
struct string_get
 {
     int        x1 ; /* ตำแหน่งในแนวแกน x มุมบนซ้ายมือของส่วนรับข้อมูล (edit)
                        ซึ่ง x1 มีค่า ตั้งแต่ 0 ถึง 89 */
     int        y1 ; /* ตำแหน่งในแนวแกน y มุมบนซ้ายมือของส่วนรับข้อมูล (edit)
                        ซึ่ง y1 มีค่า ตั้งแต่ 0 ถึง 24 */
     int       num ; /* ขนาดความยาวของ dialog box ในหน่วยตัวอักษร ซึ่งต้องมี
                        ค่าไม่เกิน 80 ตัวอักษร */
     int      mode ; /* โหมดการแก้ไขข้อมูล มี 2 โหมด คือ
                        1. โหมดแทรกข้อมูล (insert)      ให้ใส่ค่าเป็น INS
                        2. โหมดทับข้อมูลเดิม (overwrite) ให้ใส่ค่าเป็น OVR */
    char str[256] ; /* ตัวแปรสำหรับเก็บข้อมูลที่รับเข้ามา ซึ่งมีขนาดสูงสุด 256 ตัว
                        อักษร */
 } ;
```

โครงสร้างของส่วนระบบรับข้อมูล (dialog box)

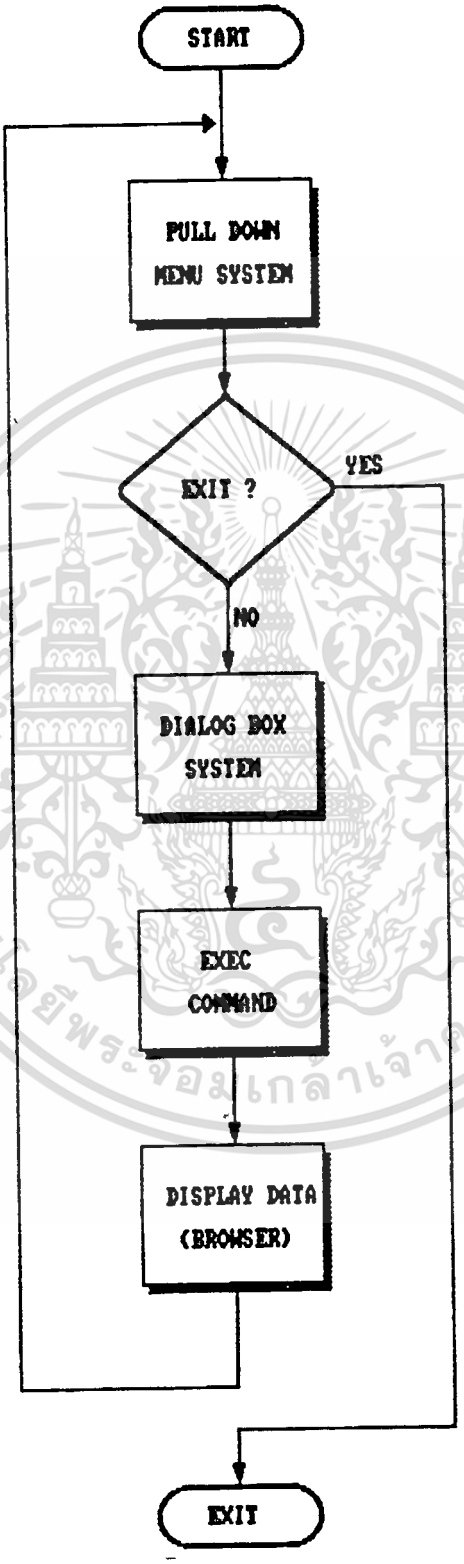
โครงสร้างของส่วนระบบรับข้อมูล (dialog box) นี้เป็นส่วนหลักของการใช้งาน ซึ่งจะรวมโครงสร้างของส่วนเลือกข้อมูล (select) ไว้ภายในด้วย โดยมีลักษณะดังนี้

```
struct dia_box
  {
    int       x1 ; /* ตำแหน่งในแนวแกน x มุมบนซ้ายมือของ dialog box ซึ่งมีค่า
                      ได้ตั้งแต่ 0 ถึง 89 */
    int       y1 ; /* ตำแหน่งในแนวแกน y มุมบนซ้ายมือของ dialog box ซึ่งมีค่า
```

```
                ได้ตั้งแต่ 0 ถึง 24 */
    int     wd ; /* ขนาดความยาวของ dialog box ซึ่งมีหน่วยเป็นตัวอักษร มีค่า
                ได้ตั้งแต่ 0 ถึง 89 */
    int     ht ; /* ขนาดความกว้างของ dialog box ซึ่งมีหน่วยเป็นตัวอักษร มีค่า
                ได้ตั้งแต่ 0 ถึง 20 */
    int  no_inp ; /* จำนวนส่วนรับข้อมูล (edit)     ที่มีอยู่ใน dialog box */
    int  no_onf ; /* จำนวนส่วนเลือกข้อมูล (select) ที่มีอยู่ใน dialog box */
    int *on_onf ; /* ที่เก็บข้อมูลของส่วนเลือกข้อมูล (select) */
    char **name ; /* ตัวแปรเก็บภาพ[illegible] dialog box */
    unsigned char **buf ;
                /* ตัวแปรเก็บรายละเอียดบนจอภาพเดิม ในส่วนที่ dialog box
                วาดทับ */
    struct string_get **string_var ;
                /* ที่เก็บข้อมูลของส่วนรับข้อมูล (edit) */
} ;
```

ขั้นตอนการทำงานของ ระบบ SHELL

จากที่ได้กล่าวมาในข้างต้นแล้วว่า ระบบ shell ประกอบด้วย ส่วนใดบ้าง สำหรับขั้นตอนการทำงานของระบบ ก็เกิดจากการนำส่วนต่างๆเหล่านั้นมารวมกัน ซึ่งสามารถแสดงโฟลว์ชาร์ตได้ดังรูป 7.1



รูป 7.1 โฟลว์ชาร์ตแสดงการทำงานของระบบ SHELL

บทที่ 8
โปรแกรมประยุกต์ใช้งานอื่นๆ

ในบทนี้จะกล่าวถึงโปรแกรมประยุกต์ใช้งานอื่นๆ ที่อำนวยประโยชน์ในการใช้งาน โดยมีทั้งโปรแกรมที่เขียนเพิ่มขึ้น และโปรแกรมที่แก้ไข โดยจะกล่าวถึงความสามารถและวิธีการใช้งานเป็นหลัก โดยโปรแกรมประยุกต์ใช้งานอื่นๆที่จะกล่าวถึงมีดังนี้

1) โปรแกรมอีดิตไฟล์แบบทีละบรรทัด : ed
2) โปรแกรมเซตสีตัวอักษรและสีพื้น : color
3) โปรแกรมขยายรหัสแทบ (tab) เป็นรหัสช่องว่าง (space bar) : prep
4) โปรแกรมแสดงเอกสารภาษาไทย/อังกฤษด้วยระบบหน้าต่าง : browser

โปรแกรมอีดี

อีดี เป็นเอดิเตอร์แบบไลน์อิดิเตอร์ (line editor) และมีขนาดเล็กมาก ดังนั้น ประสิทธิภาพ และความสามารถต่าง ๆ จึงไม่อาจเทียบเท่ากับ มินอิดิต หรือ แอลเล่ได้ อีกทั้งการใช้งานก็ยุ่งยาก ไม่สะดวก แต่เนื่องจากอีดีมีขนาดเล็ก และมีซอร์สโค้ดมาให้ด้วย ทำให้สามารถดัดแปลงโปรแกรมให้สามารถทำงานกับภาษาไทยได้ ดังนั้น เราจึงจะมาพูดถึงวิธีการใช้งานของอีดีกันสักเล็กน้อย

ก่อนที่จะอธิบายถึงวิธีใช้งานอีดีนั้น จะอธิบายถึงสาเหตุที่เอดิเตอร์ตัวอื่นไม่สามารถทำงานกับภาษาไทยที่พัฒนาขึ้นใหม่ได้ คือ ตามทฤษฎีแล้ว โปรแกรมปรกติทั่ว ๆ ไปน่าจะสามารถทำงานกับภาษาไทยได้เลย (ในที่นี้ หมายถึง โหมดแสดงผลแบบ 25 บรรทัดเท่านั้น เพราะโหมด 17 บรรทัดนั้น สำหรับเอดิเตอร์แบบฟูลสกรีน จำเป็นต้องแก้ไขโปรแกรมอยู่แล้ว) แต่ในเอลเล่ ซึ่งจะแสดงผลภาษาไทยเป็นตัวอักษรขยะ และในมินอิดิต ซึ่งไม่รับภาษาไทยเลยนั้น คาดว่าน่าจะมีการเขียนโปรแกรมเช็ครหัสของตัวอักษรอยู่ ทำให้ตัวโปรแกรมไม่รู้ว่าภาษาไทยก็เป็นตัวอักษร ซึ่งในกรณีของเอลเล่นั้น ไม่มีซอร์สโค้ดมาให้ ทำให้ไม่สามารถทำการแก้ไขได้ ส่วนมินอิดิตนั้น

ซอร์สโค้ดมีขนาดใหญ่มาก ทำให้ยากแก่การแก้ไข ดังนั้นอีดี จึงเป็นทางออกที่ดีที่สุด เนื่องจากซอร์สโค้ดมีขนาดเล็ก ทำให้การแก้ไขทำได้ง่าย ซึ่งก็ตรงตามความคาดหมาย การแก้ไข ทำเพียงที่เดียว อีดีก็สามารถทำงานกับภาษาไทยได้ทั้งสองโหมด คือ ทั้ง 17 และ 25 บรรทัด (บริเวณที่ทำการแก้ไข สามารถดูได้จากซอร์สโค้ด โดยค้นหาคำว่า THAI)

เริ่มต้นใช้งาน

การใช้งานอีดี ทำได้โดยคีย์คำสั่งดังรูป

```
$ ed prog.c
```

หมายถึง ต้องการเขียนโปรแกรมชื่อ prog.c อีดีจะทำการโหลดไฟล์เข้าสู่หน่วยความจำ โดยจะแสดงจำนวนบรรทัด และขนาดไฟล์ให้เห็น หลังจากนั้น จะรอรับคำสั่ง โดยตัวอีดีนี้ จะไม่มีคอมมานด์พร้อมต์

คำสั่งทั้งหมดของอีดี มีดังนี้คือ

(.)a: ย่อมาจาก append หมายถึงทำการเขียนเพิ่ม

(.,.)c: ย่อมาจาก change

(.,.)d: ย่อมาจาก delete หมายถึงลบบรรทัด

e: มาจาก edit new file หมายถึง จะทำการเขียนไฟล์ใหม่

f: แสดงชื่อไฟล์ปัจจุบันที่กำลังเขียนอยู่

(1,$)g: ย่อมาจาก global command

(.)i: ย่อมาจาก insert หมายถึงต้องการแทรกบรรทัด

(.,.+1)j: ย่อมาจาก join หมายถึงการต่อบรรทัดเข้าด้วยกัน

(.)k: ทำการมาสค์

(.)l: นิมพ์ข้อมูล โดยแสดงตัวอักษรนิเศษเป็นเลขฐานแปด

(.,.)m: ย่อมาจาก move หมายถึง การย้ายข้อมูล

(...)p: ย่อมาจาก print หมายถึง พิมพ์ข้อมูล

q: ย่อมาจาก quit หมายถึง เลิกทำงาน กลับไปที่พร้อมต์

(.)r: ย่อมาจาก read หมายถึง อ่านไฟล์จากแผ่นดิสก์ เข้าสู่หน่วยความจำ

(...)s: ย่อมาจาก substitute หมายถึง แทนที่คำ

(1,$)v: ทำงานเหมือนคำสั่ง g ยกเว้นบรรทัดที่ไม่แมทช์กับที่ได้เลือกไว้

(1,$)w: ย่อมาจาก write หมายถึง เขียนไฟล์ลงแผ่นดิสก์

คำสั่งส่วนใหญ่ จะต้องใส่นารามิเตอร์ซึ่งเป็นหมายเลขบรรทัดหนึ่ง หรือสองตัว ถ้าไม่ใส่ จะมีความหมายตามที่แสดงค่าไว้ในวงเล็บ โดย จุด หมายถึงบรรทัดปัจจุบัน และเครื่องหมายดอลล่าร์ หมายถึงบรรทัดสุดท้ายของไฟล์ เช่น คำสั่ง d ถ้าไม่มีพารามิเตอร์ จะหมายถึงลบบรรทัดปัจจุบัน

ตัวอย่างการใช้งานคำสั่ง เช่น

3,20p

หมายถึง ให้พิมพ์บรรทัดที่ 3 ถึงบรรทัดที่ 20

/whole/

หมายถึง ให้ค้นหาคำว่า 'whole'

s/whole/while/

หมายถึง ต้องการค้นหา และแทนที่คำว่า 'whole' ด้วยคำว่า 'while'

g/MAXBUF/s//MAX_BUF/g

หมายถึง ต้องการค้นหา และแทนที่คำว่า 'MAXBUF' ด้วยคำว่า 'MAX_BUF' ทุกที่ตลอดทั้งไฟล์

อนึ่ง สำหรับคำสั่ง a นั้น เมื่อใช้คำสั่งนี้ อีดีจะอยู่ในโหมดเขียนเพิ่ม โดยจะรับข้อความไปเรื่อย ๆ เพิ่มเข้าไปในไฟล์ การจะกลับเข้าสู่โหมดรับคำสั่งอีก จะทำได้โดยพิมพ์จุดที่คอลัมภ์แรก และจะต้องเป็นตัวอักษรตัวเดียวในบรรทัดเท่านั้น เมื่อกดปุ่มรีเทริน์ อีดีก็จะกลับเข้าสู่โหมดรับคำสั่ง ซึ่งผู้ใช้ต้องรู้เอง เนื่องจากอีดีไม่มีพร้อมต์รับคำสั่ง

สำหรับคำสั่ง r นั้น เป็นการอ่านไฟล์จากแผ่นดิสก์ เข้าสู่หน่วยความจำ แต่จะไม่เคลียร์ไฟล์เก่าถ้าหากมี กล่าวคือ ถ้าทำการเขียนไฟล์หนึ่ง แล้วใช้คำสั่ง w เพื่อเขียนไฟล์ลงดิสก์ เพื่อจะใช้คำสั่ง r อ่านไฟล์อีกไฟล์ขึ้นมาเขียน จะทำให้ในหน่วยความจำ จะประกอบด้วยไฟล์สองไฟล์ต่อกัน คือ ถ้าใช้คำสั่ง w เพื่อเขียนไฟล์ลงแผ่น จะได้ข้อมูลของทั้งสองไฟล์รวมกัน ดังนั้น ทุกครั้งที่จะทำการเขียนไฟล์ใหม่ จะต้องใช้คำสั่ง q เลิกการทำงาน แล้วจึงค่อยเรียกโปรแกรมอีดิใหม่อีกครั้ง

โปรแกรม color

โปรแกรม color เป็นโปรแกรมที่ใช้ในการเซตสีตัวอักษรและสีพื้น (foreground and background color) ทำงานโดยรับหมายเลขสีจากคอมมานด์ไลน์ และสร้างรหัสควบคุม (escape sequences) สำหรับเซตสีตัวอักษรและสีพื้น แล้วจึงจบการทำงาน โดยมีการเรียกใช้งาน ดังนี้

```
color #foreground_num [#background_num]
```

ในการเรียกใช้งานโปรแกรม color สามารถกำหนดเพียงสีตัวอักษรเพียงอย่างเดียว หรือ กำหนดทั้งสีตัวอักษรและสีพื้นพร้อมกันก็ได้ เช่น color 2 , color 4 3 , color 7 0 เป็นต้น

โปรแกรม prep

เป็นโปรแกรมที่ใช้ในการ กำจัดรหัสควบคุมพิเศษอื่นๆออก เพื่อให้ไฟล์เหลือแต่เพียงข้อความภาษาอังกฤษ/ไทยเท่านั้น นอกจากนี้ยังทำการขยายรหัสแทบ (tab character) ออกเป็นรหัสช่องว่าง (space bar) โดยกำหนดจำนวนช่องว่างต่อหนึ่งรหัสแทบได้ โปรแกรมนี้เหมาะสำหรับแปลงไฟล์เอกสารที่ได้จากโปรแกรมเวิร์ดโปรเซสเซอร์ภาษาไทย (Thai Word Processor) ที่ใช้บนเอ็มเอสดอสทั่วไป ให้มาใช้บนมินิกซ์ เช่น นำไปใช้ในโปรแกรมแสดงเอกสารภาษาไทย/อังกฤษด้วยระบบหน้าต่าง (โปรแกรม browser) เป็นต้น

รูปแบบการใช้งานของโปรแกรม prep มีดังนี้

```
prep inputfile outputfile [#tabsize]
```

โดย inputfile หมายถึงชื่อไฟล์ที่เก็บเอกสารแบบเท็กซ์ไฟล์

outputfile หมายถึงชื่อไฟล์ที่ผ่านการตัดรหัสพิเศษออก

tabsize หมายถึงขนาดของรหัสแทบ(ถ้าไม่บอกจะกำหนดให้มีค่าเป็น 8)

ตัวอย่างเช่น

```
prep document.cw document
```

โปรแกรม browser

เป็นโปรแกรมที่มีความสามารถในการแสดงเอกสารทั้งภาษาไทยและภาษาอังกฤษบนระบบหน้าต่างที่มีขนาดเปลี่ยนแปลงได้ (variable-sized window system) โดยสามารถกวาดดูเอกสารกลับไปกลับมาได้ สามารถเลื่อนเอกสารในหน้าต่างได้ทั้งในแนวนอนและแนวตั้งอย่างอิสระ และใช้ได้กับเอกสารขนาดเท่าใดก็ได้ เนื่องจากการแสดงเอกสารจะอ่านจากดิสค์ด้วยวิธีการอ่านเมื่อจำเป็น (page demand) โดยสามารถกำหนดขนาดของบัฟเฟอร์ได้ตั้งแต่ขนาด 8K ถึง 31K โดยการแก้ไขที่โปรแกรม สำหรับขนาดของหน้าต่างสามารถเซตได้เมื่อทำการเรียกใช้ สำหรับเอกสารที่ต้องการจะแสดง ควรจะเป็นเอกสารที่ผ่านการทำตัดรหัสพิเศษออกด้วยโปรแกรม prep แล้ว

การเรียกใช้โปรแกรม browser ทำได้โดยเรียกใช้

```
browser filename "Logo Name" [#x1 #y1 #xr #yr]
```

โดย filename หมายถึงชื่อไฟล์ที่เก็บเอกสารที่ต้องการจะแสดง

"Logo Name" หมายถึงชื่อหัวข้อที่จะแสดงบนส่วนหัวของระบบหน้าต่าง

และ #x1 #y1 #xr #yr หมายถึงตำแหน่งมุมบนซ้ายและมุมล่างขวาของหน้าต่าง ซึ่งถ้าไม่กำหนด จะมีค่าปกติตั้งไว้แล้ว

ตัวอย่างเช่น

```
browser document "เอกสารประกอบการศึกษา" 1 1 80 17
```

การเรียกใช้โปรแกรม browser กับเอกสารภาษาไทย จะต้องสั่งให้โปรแกรมทำงานในระบบภาษาไทยจัดระดับ (โหมด 17 บรรทัด) ด้วย มิฉะนั้นเอกสารจะไม่จัดระดับและแสดงผลผิดพลาดได้ และระบบภาษาไทยที่สนับสนุนคือรหัสแอลกี้สมอ.

คำสั่งในขณะสั่งโปรแกรม browser ทำงาน

หลังจากสั่งให้โปรแกรม browser ทำงานแล้ว โปรแกรมจะสร้างหน้าต่างพร้อมแสดงชื่อหัวข้อของเอกสาร และทำการอ่านข้อความเข้าสู่บัฟเฟอร์และแสดงออกมาภายในหน้าต่าง โดยบรรทัดสุดท้ายจะแสดงชุดคำสั่งที่สนับสนุน โดยมีรายละเอียดดังนี้

ปุ่มลูกศรต่างๆ	เป็นการเลื่อนเอกสารไปมาตามทิศทางของลูกศร
PageUp	เป็นการเลื่อนเอกสารไปข้างหน้าหนึ่งหน้า
PageDown	เป็นการเลื่อนเอกสารถอยหลังกลับไปหนึ่งหน้า
Home	เป็นการแสดงตัวอักษรตัวแรกของบรรทัดที่คอลัมน์แรกของหน้าต่าง
End	เป็นการแสดงตัวอักษรที่อยู่ขวาสุดของหน้านั้นให้อยู่ไม่เกินคอลัมน์สุดท้ายของหน้าต่าง
Ctrl-Q	เป็นปุ่มที่ใช้ในการออกจากโปรแกรม และตอบยืนยันด้วย y
Ctrl-D	เป็นปุ่มที่ใช้ในการก็อปปี้เอกสารที่แสดงไปเก็บไว้ในไฟล์ที่มีชื่อว่า /tmp/dump (ใช้ในกรณีโปรแกรม browser ทำงานร่วมกันกับโปรแกรมเชลล์)

บทที่ 9

สรุป ปัญหาและแนวทางในการพัฒนาต่อ

สำหรับการทำโครงงานนี้ ในภาคเรียนแรกเป็นการศึกษาทฤษฎีที่ใช้ และศึกษาซอร์สโค้ด เพื่อมองถึงการจัดโครงสร้างของโปรแกรม เพื่อใช้ในการพัฒนาระบบ และได้มีการก็อปปี้ซอฟท์แวร์นี้มาใช้งาน โดยครั้งแรกได้แผ่นมินิกซ์เวอร์ชัน 1.1 ซึ่งได้มาแบบไม่สมบูรณ์ และมาได้แผ่นมินิกซ์เวอร์ชัน 1.3 ในปลายภาคการศึกษาแรกซึ่งเป็นเวอร์ชันที่มีความแตกต่างจากเวอร์ชัน 1.1 หลายประการ เช่น สามารถทำงานได้กับเครื่องไอบีเอ็มพีซี เอที 386 ได้, สามารถจองรู๊ตไฟล์ซิสเต็มในหน่วยความจำเหนือ 1 เมกกะไบต์ได้สำหรับเครื่องที่มีหน่วยความจำขยาย (extended memory) ตลอดจนเพิ่มความสามารถในการต่อระบบเป็นแบบมัลติยูสเซอร์ผ่านพอร์ต RS232 และเพิ่มคอมมานด์โปรแกรมช่วยมากยิ่งขึ้น หลังจากนั้นในภาคเรียนที่สอง เป็นการออกแบบลักษณะโครงร่างของระบบภาษาไทย ค้นหาจุดต่างๆที่ต้องทำการแก้ไข เพิ่มเติมส่วนจัดการกับภาษาไทย และเขียนโปรแกรมประยุกต์ใช้งานสำหรับทดลองการทำงานของระบบ

ในระบบปฏิบัติการมินิกซ์ได้แบ่งการทำงานออกเป็นส่วนๆที่สำคัญโดยสรุป คือ

- เคอร์นัล จะทำหน้าที่ในการควบคุมและจัดการเกี่ยวกับโปรเซส ได้แก่ การจัดสรรเวลาให้กับโปรเซส การติดต่อสื่อสารระหว่างโปรเซส และรวมทั้งการจัดการเกี่ยวกับดีไวฟ์ไดร์ฟเวอร์ต่างๆ ได้แก่ การจัดการเกี่ยวกับเทอร์มินัล แป้นพิมพ์ ดิสไดร์ฟ แรมดิสค์ รีโมทเทอร์มินัล (RS232) และเครื่องพิมพ์

- ระบบไฟล์ จะทำหน้าที่ในการจัดการเกี่ยวกับระบบไฟล์ ได้แก่ ไดเร็กตอรี ซุบเปอร์บล็อก ตารางของไอโหนด การควบคุมการเข้าถึงไฟล์ (access control) และการแยกแยะผู้ใช้ (user autheticatin) และการทำแคช

- ระบบหน่วยความจำ จะทำหน้าที่ในการจัดการส่วนที่เกี่ยวกับหน่วยความจำ ได้แก่ การวางโครงร่างของหน่วยความจำ การส่งข่าวสาร การจัดการเกี่ยวกับซิสเต็มคอลล์ที่เกี่ยวกับหน่วยความจำ การส่งสัญญาณ

ข้อดีของระบบปฏิบัติการมินิกซ์ จะมองได้สองประการใหญ่ๆ คือ ประการแรก มินิกซ์เป็นระบบปฏิบัติการที่มีความสามารถในการทำหลายๆงานพร้อมๆกัน และมีซอร์สโปรแกรมให้ศึกษา จึงเหมาะสำหรับผู้ที่ต้องการศึกษาการออกแบบและสร้างระบบปฏิบัติการ รวมไปถึงการพัฒนาต่อให้ระบบมีความสามารถยิ่งขึ้น ประการที่สอง มินิกซ์เป็นระบบปฏิบัติการที่มีการใช้งานคล้ายกันกับยูนิกซ์ ดังนั้นจึงเหมาะกับผู้ต้องการศึกษาการใช้งานระบบยูนิกซ์ แต่ไม่มีซอฟท์แวร์ยูนิกซ์ หรือขาดแคลนฮาร์ดแวร์และอุปกรณ์ที่ต้องติดตั้งเพิ่มเติมเพื่อใช้ติดตั้งระบบยูนิกซ์ เพราะมินิกซ์ใช้ได้กับระบบขนาดเล็ก นอกจากนี้ผู้ใช้ยังสามารถศึกษาการบริหารระบบ (system administration) ได้โดยไม่จำเป็นต้องเป็นเจ้าของระบบยูนิกซ์จริงๆ

สำหรับข้อเสียหรืออาจกล่าวได้ว่าเป็นจุดด้อยที่สำคัญของระบบปฏิบัติการมินิกซ์ ก็คือ ประสิทธิภาพในการทำงาน ซึ่งเป็นผลสืบเนื่องมาจากการออกแบบระบบปฏิบัติการที่มีการติดต่อกันระหว่างโปรเซสเป็นแบบการส่งข่าวสาร (message passing) ซึ่งไม่เหมาะสมกับการใช้งานบนเครื่องไมโครคอมพิวเตอร์ เพราะต้องเสียโอเวอร์เฮดมาก โปรเซสบางโปรเซสจะต้องติดต่อกับข่าวสารที่รออยู่ในคิวจำนวนมาก จะทำให้โปรเซสอื่นๆที่เป็นผู้สร้างข่าวสารต้องรอผลการทำงานนาน และการแก้ไขให้มินิกซ์เปลี่ยนไปใช้การติดต่อกันระหว่างยูสเซอร์กับโอเอสเป็นแบบการเรียกเป็นโปรซีเยอร์ (procedural call) แบบยูนิกซ์ก็เป็นไปได้ยาก เพราะต้องเปลี่ยนโครงสร้างของระบบใหม่หมด

ในส่วนของระบบปฏิบัติการมินิกซ์ ภาษาไทยได้ทำการแก้ไขและเพิ่มเติมในส่วนของเคอร์นัล ได้แก่

- แก้ไขเทอร์มินัลในส่วนของเคอร์นัล เพื่อให้สามารถแสดงผลภาษาไทยที่เป็นกราฟิกโหมด
- เพิ่มโปรเซสที่เกี่ยวกับระบบภาษาไทย เพื่อใช้บริการแก่ผู้ใช้โดยจัดทำเป็นซิสเต็มคอลล์ เช่น การอ่านตัวอักษรบนหน้าจอ การเปลี่ยนโหมดภาษาไทยแบบจัดและไม่จัดระดับ เป็นต้น
- เพิ่มลักษณะของตัวอักษรลงในเคอร์นัล โดยมี 2 รูปแบบ คือขนาดตัวอักษร 8x14 และขนาด 8x20 จุด แต่ละแบบจะมีให้ 2 ชุดให้ใช้ โดยสามารถติดตั้งฟอนต์เพิ่มได้
- เพิ่มซอฟท์แวร์เคอร์เซอร์ ซึ่งจะทำการกระพริบทุกๆครึ่งวินาที
- เพิ่มโปรแกรมประยุกต์ใช้งานบนมินิกซ์ซึ่งเป็นเชลล์ในการจัดการกับระบบ เช่น การก็อปปี้ไฟล์ การลบไฟล์ การแสดงไฟล์ การสร้างและลบไดเรกทอรี และการจัดการกับดิไวซ์ต่างๆ

โดยมีการใช้งานเป็นแบบเมนู

- มีปุ่มคีย์ในการเลือกโหมดภาษาไทย/ภาษาอังกฤษ และลักษณะการแสดงผล
- แก้ไขเชลล์ให้สามารถรับข้อความภาษาไทยได้

ข้อดีของระบบภาษาไทยบนมินิกซ์ ได้แก่

- เป็นระบบภาษาไทยที่ไม่ขึ้นกับฮาร์ดแวร์
- สามารถแสดงผลตัวอักษรได้หลายๆแบบพร้อมๆกัน
- สามารถวาดรูปพร้อมๆกับแสดงตัวอักษรพร้อมๆกันได้

ข้อเสียของระบบภาษาไทยบนมินิกซ์ ได้แก่

- ในการแสดงผลระบบภาษาไทยบนกราฟิกโหมด จะทำให้การทำงานที่เกี่ยวกับจอภาพช้า
- ต้องเสียเวลาในการวาดเคอร์เซอร์ใหม่เพราะกระทำการด้วยซอฟท์แวร์
- เนื่องจากส่วนของเคอร์นัลมีขนาดใหญ่จึงทำให้หน่วยความจำที่เหลือใช้มีขนาดน้อยลง
- ต้องเสียเนื้อที่ที่ใช้ในการเก็บรูปแบบของตัวอักษร

ส่วนปัญหาในการทำมินิกซ์นี้

- ในโปรแกรมระบบเดิม ไม่สามารถรับข้อมูลที่เป็นภาษาไทยได้ เช่น อิดิเตอร์ จึงทำให้ต้องเขียนโปรแกรมที่ต้องใช้ภาษาไทยในระบบปฏิบัติการของดอสก่อน แล้วค่อยก๊อปปี้โปรแกรมลงในระบบปฏิบัติการของมินิกซ์
- ในการแก้ไขโปรแกรมกระทำได้ลำบาก เพราะขาดโปรแกรมที่ช่วยในการดีบักโปรแกรม
- การสร้างแผ่นบูตใหม่จะต้องใช้เครื่องที่มีหน่วยความจำและเนื้อที่ดิสค์มาก
- ในการคอมไพล์โปรแกรม ถึงแม้โปรแกรมจะมีขนาดเล็ก ก็ใช้เวลาในการคอมไพล์นาน เพราะคอมไพเลอร์ยังมีประสิทธิภาพค่อนข้างต่ำและขาดการพัฒนาต่อ

สำหรับแนวทางในการพัฒนาต่อ น่าจะเป็นดังนี้

- ควรจัดให้มีการทดลองติดต่อสื่อสารระหว่างเครื่องคอมพิวเตอร์ โดยผ่านทาง RS-232

เพื่อทำเป็นระบบหลายผู้ใช้

- ควรจัดให้มีการทดลองใช้มินิกซ์กับอาร์คดิสค์
- เพิ่มโปรแกรมประยุกต์ใช้งานภาษาไทยอื่นๆ เช่น โปรแกรมสำนักพิมพ์ตั้งโต๊ะ(desktop publishing) เป็นต้น ให้มากขึ้น
- เปลี่ยนระบบการจัดการหน่วยความจำให้เป็นแบบ segmentation หรือ paging เพื่อทำเป็นระบบที่มีความปลอดภัย

ภาคผนวก ก.

กราฟฟิคโหมดบนจอภาพอีจีเอ/วีจีเอและเฮอร์คิวลิส

สถาปัตยกรรมของจอภาพ EGA

สถาปัตยกรรมพื้นฐานของจอภาพอีจีเอ จะประกอบไปด้วยส่วนต่างๆ 6 ส่วนด้วยกัน คือ

1. Display memory เป็นหน่วยความจำแบบไดนามิกแรมขนาด 256 กิโลไบต์ โดยแบ่งออกเป็นสี่ระนาบสี ใช้สำหรับเก็บข้อมูลกราฟฟิคบนจอภาพ

2. Graphics Controller จะอยู่ระหว่างทางเดินของข้อมูลและหน่วยความจำจอภาพสามารถโปรแกรมใช้ในการทำฟังก์ชันทางโลจิก เช่น AND, OR, XOR ,ROTATE กับข้อมูลที่จะเขียนลงในหน่วยความจำจอภาพ

3. CRT Controller เป็นตัวสร้างสัญญาณนาฬิกาเพื่อที่จะควบคุมการทำงานของจอภาพและกำหนดเวลาในการทำรีเฟล็กช (refresh) จอภาพ

4. Data Serializer จะทำการอ่านข้อมูลจากหน่วยความจำจอภาพ และแปลงข้อมูลออกเป็นบิตสตรีมเพื่อที่จะส่งไปที่จอภาพ

5. Attribute Controller ประกอบไปด้วยตารางแมปสี(Color Look-up Table) ซึ่งจะใช้ในการเลือกสี 16 สีจากสีทั้งหมด 64 สี

6. Sequencer ทำหน้าที่ในการควบคุมสัญญาณเวลาของฟังก์ชันทั้งหมด รวมทั้งลอจิกในการเลือกระนาบสีที่จะประมวลผล

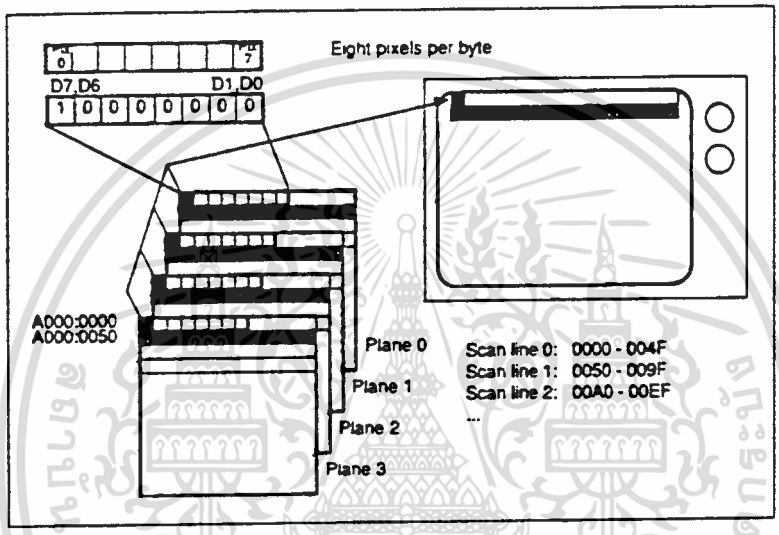
กราฟฟิคโหมดบนจอภาพอีจีเอ

จอภาพอีจีเอสามารถแสดงผลในโหมดกราฟฟิคด้วยความละเอียด 640 x 350 จุด โดยใช้ระนาบสีทั้งสี่ระนาบที่เป็นอิสระต่อกัน ในแต่ละจุดจะใช้หนึ่งบิตในแต่ละระนาบสี ดังนั้นจุดหนึ่งจุดสามารถแสดงสีได้เท่ากับ 16 สี ในการคำนวณแอดเดรสของหน่วยความจำจอภาพเพื่อที่จะใช้ในการเขียนจุด x,y ลงบนจอภาพสามารถคำนวณได้จากสมการ

ออฟเซต = y * 80 + x/8

ตำแหน่งบิต = 7 - (x modulus 8)

โดย x เป็นพิกัดในขอบเขตระหว่าง 0 - 639 และ y อยู่ในขอบเขตระหว่าง 0 - 349 สำหรับเซกเมนต์ของวีดีโอแรมจะอยู่ที่ 0xA000



รูป ก.1 แสดงการแมปหน่วยความจำสำหรับโหมดกราฟฟิคบนจออีจีเอ

ในการติดต่อกับหน่วยความจำจอภาพ ทั้งการอ่านและเขียนข้อมูล สามารถทำได้หลายวิธีด้วยกัน ซึ่งแต่ละวิธีจะมีวิธีโปรแกรมอีจีเอรีจิสเตอร์ต่างกัน ดังนี้

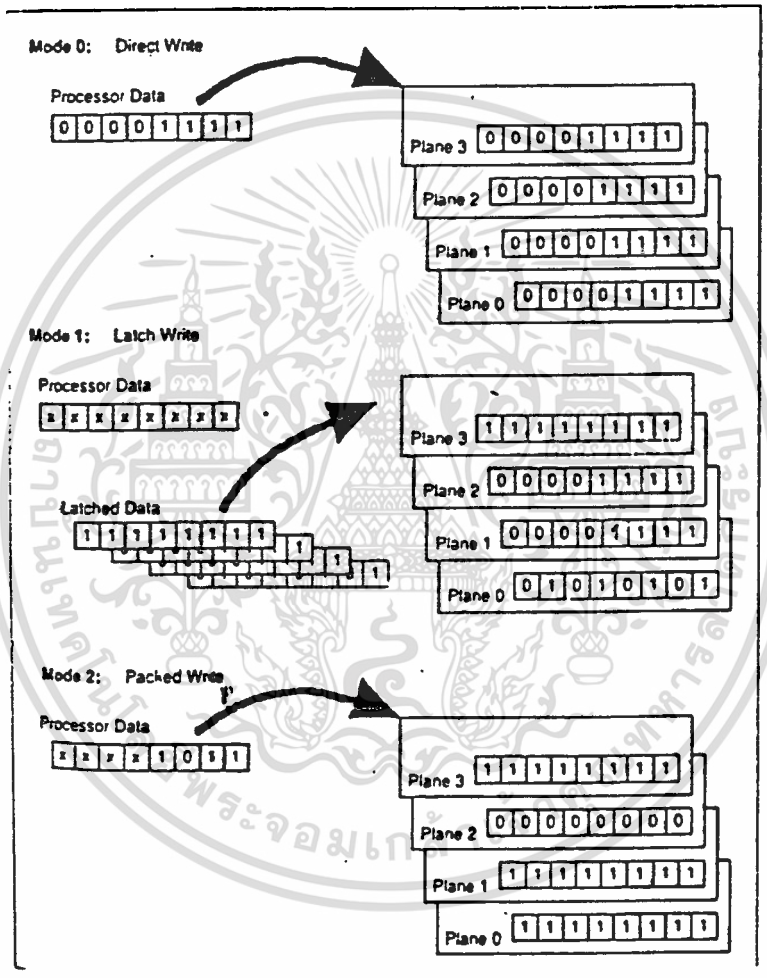
1. Processor Write using Set-Reset จะทำการเขียนแพทเทอร์นสีที่กำหนดอยู่ในรีจิสเตอร์ Set/Reset ลงในหน่วยความจำจอภาพ ในโหมดนี้สามารถเขียนข้อมูลได้มากที่สุดถึง 8 จุดพร้อมๆกัน โดยการเลือกระนาบสีทุกระนาบ และเหมาะสมกับการเขียนจุดหนึ่งจุด ด้วยอัลกอริทึม DDA (Digital Differential Analyzer) นอกจากนี้ยังใช้ในการวาดเส้นตรงอีกด้วย

2. Processor Write using Planar Pixel Write เป็นการเขียนจุดข้อมูลเป็นระนาบไปยังหน่วยความจำจอภาพ โดยในการเลือกระนาบที่จะทำการเปลี่ยนแปลงสามารถทำได้ผ่านรีจิสเตอร์ Write Plane Enable และ เลือกบิตที่จะแก้ไขด้วยการเซตมาร์ค (Mask)

รีจิสเตอร์ การเขียนข้อมูลลงในจอภาพในโหมดนี้เหมาะสมสำหรับการโหลดข้อมูลภาพเป็นระนาบครั้งละหนึ่งระนาบ เหมาะกับการโหลดภาพที่จัดเก็บไว้แสดงบนจอภาพ

3. Latched Data Write จะเขียนข้อมูลที่แลทช์ไว้โดยโปรเซสเซอร์ลงในหน่วยความจำจอภาพ โดยในแต่ละระนาบนั้นโปรเซสเซอร์จะมีแลทช์ของตัวมันเองโดยจะเก็บข้อมูลที่ถูกอ่านครั้งล่าสุด วิธีนี้เหมาะสำหรับการก็อปปี้ข้อมูลจากตำแหน่งหนึ่งบนจอภาพไปยังอีกตำแหน่งของจอภาพ โดยในแต่ละครั้งจะทำการก็อปปี้ข้อมูลได้ถึง 32 บิต (8 บิตต่อหนึ่งระนาบ)

4. Packed Pixel Write เขียนข้อมูลจุดชนิด packed จากโปรเซสเซอร์ไปยังหน่วยความจำจอภาพ โดยข้อมูลสี่บิตล่างที่เก็บในโปรเซสเซอร์จะเขียนลงในระนาบทั้งสี่ หรืออาจจะกล่าวได้ว่าข้อมูลที่เก็บในโปรเซสเซอร์เปรียบเสมือนสีของจุดๆนั้น ในแต่ละครั้งสามารถเขียนจุดได้ตั้งแต่ 1 ถึง 8 จุดขึ้นอยู่กับค่าที่เก็บในมาร์ครีจิสเตอร์ วิธีนี้เหมาะในการโหลดภาพที่เก็บอยู่ในฟอร์แมทที่แพ็คไว้ เช่น ไฟล์ข้อมูลภาพ TIFF เป็นต้น



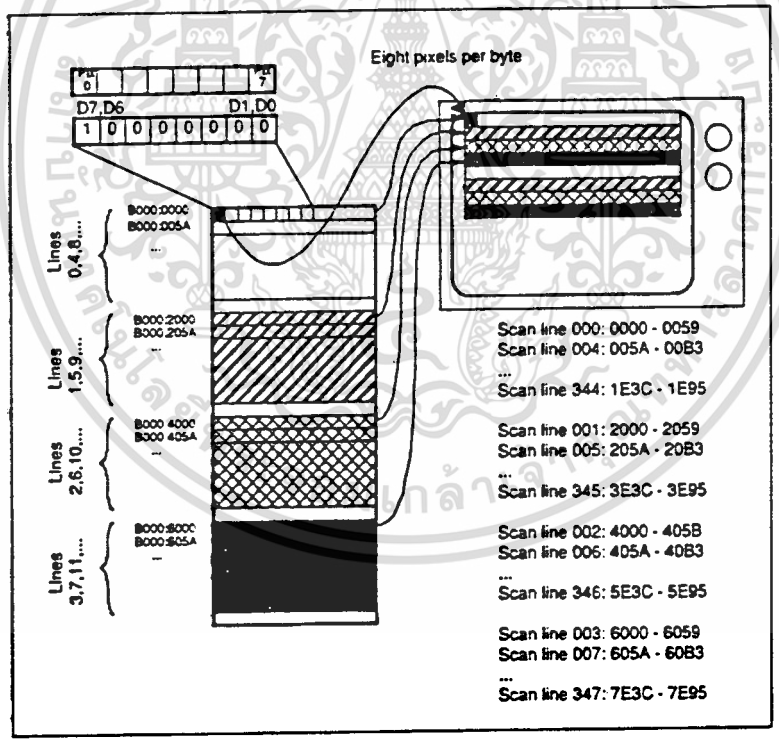
รูป ก.2 แสดงโหมดในการเขียนจุด

กราฟิคของจอเฮอร์คิวลิส

สำหรับจอภาพเฮอร์คิวลิส เป็นจอภาพที่สามารถแสดงผลได้สองสีและมีความละเอียดในการแสดงจุดทั้งหมด 720 x 348 การแสดงผลทางกราฟิคจะเป็นแบบบิตแมปเช่นเดียวกันกับจอภาพอีจีเอ/วีจีเอ แต่ละจุดจะแมปไปยังวีดีโอแรมหนึ่งบิต โดยตำแหน่งของวีดีโอแรมจะอยู่ที่เซกเมนท์0xB000สำหรับเพจศูนย์และเซตเมนต์0xB800 สำหรับเพจหนึ่ง และมีตำแหน่งออฟเซตในการกำหนดจุดคำนวณได้ดังนี้

ออฟเซต = ((y%4) * 0x2000) + (90 * (y/4)) + x/8

ตำแหน่งบิต = 7 - (x % 8)



รูป ก.3 แสดงการแมปหน่วยความจำในโหมดกราฟิคสำหรับจอเฮอร์คิวลิส

ภาคผนวก ข.

การใช้ระบบปฏิบัติการมินิกซ์

การใช้งานระบบปฏิบัติการมินิกซ์ อย่างน้อยที่สุดจะต้องมีคอมพิวเตอร์ไอบีเอ็มพีซี/เอ็กซ์ที หรือพวกที่ compatible กันก็ได้ เพราะไม่ใช่คอมพิวเตอร์ทุกเครื่องที่ compatible แล้วจะใช้งานมินิกซ์ได้ ซึ่งขึ้นอยู่กับผู้ผลิตด้วย เนื่องจากมินิกซ์ไม่ใช้ไบออส (เช่นเดียวกันกับระบบยูนิกซ์) จึงต้องควบคุมอุปกรณ์ภายนอก (i/o chips) โดยตรง ทางที่จะทดสอบว่ามินิกซ์สามารถทำงานได้หรือไม่ ก็ต้องลองใช้กับเครื่องนั้นๆดูว่าใช้ได้หรือไม่

มินิกซ์มีหลายเวอร์ชั่น ต่างต้องการลักษณะเครื่องคอมพิวเตอร์ที่แตกต่างกัน ซึ่งอย่างน้อยมินิกซ์จะใช้งานได้ ต้องมีคุณสมบัติดังนี้

1. มีหน่วยความจำอย่างน้อย 512K
2. มีดิสค์ไดรฟ์ 360 K อย่างน้อย 1 ตัว

สำหรับการใช้งานปกติเครื่องคอมพิวเตอร์ควรมีหน่วยความจำอย่างน้อย 640 K และ มีดิสค์ไดรฟ์ 360 K จำนวน 2 ตัว สำหรับการใช้ระบบปฏิบัติการมินิกซ์สำหรับพัฒนาระบบควรจะเป็นเครื่องไอบีเอ็ม เอที 386 และมีหน่วยความจำอย่างน้อย 2 เมกกะไบต์ จึงจะมีประสิทธิภาพมากที่สุด นอกจากนี้มินิกซ์ยังสามารถใช้ร่วมกับฮาร์ดดิสค์ได้ด้วย ซึ่งสามารถหารายละเอียดการติดตั้งฮาร์ดดิสค์ลงในระบบได้จากเอกสาร (document) ในแผ่นมินิกซ์

เมื่อเป็นผู้ใช้ระบบ

ก่อนจะเริ่มใช้งานระบบมินิกซ์ควรทำ back up ไว้ก่อน จากนั้นจึงเริ่ม boot ระบบมินิกซ์ตามขั้นตอนดังนี้

1. เปิดเครื่องคอมพิวเตอร์ แล้วใส่แผ่นดิสค์ชื่อ boot diskette ในไดรฟ์ A จากนั้นจะปรากฏข้อความว่า "Booting Minix 1.3" หลังจากนั้นเครื่องคอมพิวเตอร์จะทำการอ่านตัวเคอร์นัลลงในหน่วยความจำ

2. ประมาณ 15 วินาทีหลังจากได้รับข้อความแล้ว จะปรากฏเมนู ซึ่งมีหัวข้อให้เลือกการทำงานต่างๆ ให้เปลี่ยนแผ่นดิสค์ ในไดรฟ์ A เป็นแผ่นดิสค์ชื่อ "Root File

system" แล้วกดคีย์ "="

3. จากนั้นมินิกซ์จะเคลียร์จอภาพ และ ที่บรรทัดแรกของจอภาพ จะปรากฏข้อความแสดงจำนวนหน่วยความจำที่มีอยู่ ขนาดของระบบมินิกซ์ จำนวนแรมดิสค์ จำนวนหน่วยความจำที่เหลือไว้ใช้งาน

4. ในขณะนี้ มินิกซ์กำลัง copy root file system จากไดรฟ์ A ไปยังแรมดิสค์

5. เมื่อแรมดิสค์ถูกโหลดเป็นที่เรียบร้อยแล้ว โปรแกรม system initialization ซึ่งอยู่ในไฟล์ /etc/rc จะทำงานต่อ และ สั่งให้เปลี่ยนแผ่นดิสค์ในไดรฟ์ A เป็นแผ่นดิสค์ชื่อ user file system (/usr) แล้วให้กดคีย์ enter

6. จากนั้นมินิกซ์จะให้ใส่วัน เวลา โดยใส่เป็นตัวเลขจำนวน 12 ตัว ตามแบบฟอร์มดังนี้ MMDDYYhhmmss ตัวอย่าง สมมติ 3:35 pm วันที่ 4 กรกฎาคม ค.ศ. 1976 ให้ใส่ข้อมูลเป็น 070476153500

7. จากนั้นจะปรากฏข้อความดังนี้ "login : " ให้พิมพ์ "ast" จากนั้นมินิกซ์ จะถาม password ให้พิมพ์ "Wachtwoord" สำหรับตัวอักษรใหญ่ และ เล็กนั้นต่างกันให้ระวังด้วย

8. เมื่อ log in ได้แล้ว จะปรากฏ prompt บนจอภาพ เป็นรูป # ซึ่งก็เป็นการเสร็จสิ้นการเข้าสู่ระบบมินิกซ์

9. เมื่อใช้งานระบบมินิกซ์เสร็จเรียบร้อยแล้ว ให้กดคีย์ "CTRL D" จะปรากฏข้อความ "login : " ถ้าต้องการใช้งานต่อให้ปฏิบัติเหมือนขั้นตอนที่ 7.

หมายเหตุ

1. แผ่นบู๊ตดิสค์มีอยู่ด้วยกัน 3 แผ่นด้วยกัน คือ แผ่นแรก จะเป็นแผ่นบู๊ตสำหรับระบบปฏิบัติการมินิกซ์เดิม (เวอร์ชัน 1.3) ใช้สำหรับการใช้ระบบเดิม แผ่นที่สอง จะเป็นแผ่นบู๊ตสำหรับระบบมินิกซ์ภาษาไทยสำหรับจอภาพอีจีเอ/วีจีเอ และแผ่นสุดท้าย เป็นแผ่นบู๊ตสำหรับระบบมินิกซ์ภาษาไทยสำหรับจอเฮอร์คิวลิส สำหรับแผ่นอื่นๆที่เหลือสามารถใช้ร่วมกันได้ ไม่ว่าจะบู๊ตด้วยแผ่นใด

2. นอกจากยูสเซอร์ ast แล้วในระบบมินิกซ์ภาษาไทยยังมียูสเซอร์ชื่อ "วิ", "เอ",

"จิ๋ว" อยู่ด้วย โดยการป้อนข้อความภาษาไทยให้กดปุ่ม F10 ก่อนเพื่อเปลี่ยนโหมดแป้นพิมพ์จากภาษาอังกฤษเป็นภาษาไทย และเคอร์เซอร์จะเปลี่ยนรูปร่าง สำหรับ password จะใช้ชื่อเดียวกันกับชื่อยูสเซอร์ หลังจากป้อนข้อความภาษาไทยเสร็จแล้วให้กดปุ่ม F10 อีกครั้งก็จะกลับไปเป็นโหมดภาษาอังกฤษเหมือนเดิม

3. การใช้งานมินิกซ์จะมีลักษณะคล้ายคลึงกันกับยูนิกซ์มาก สามารถใช้คำสั่งมาตรฐานของยูนิกซ์ได้ เช่น cp, ls, cd และยังมีตัวอักษรพิเศษที่ใช้ควบคุม (control character) เช่น '@' หมายถึงยกเลิกตัวอักษรที่ป้อนเข้าไปบรรทัดนั้น (kill line), Ctrl-S หมายถึงหยุดการทำงานของระบบ และจะทำงานต่อเมื่อมีการกด Ctrl-Q เป็นต้น

4. มินิกซ์ใช้ปุ่ม Ctrl-\ ในการ kill โปรเซสขณะนั้น และใช้ปุ่ม Ctrl-F9 ในการ kill ทุกโปรเซสที่ทำงานอยู่ และกลับไปสู่การ login ใหม่อีกครั้ง ซึ่งปุ่มนี้จะใช้เมื่อปุ่ม Ctrl-\ ใช้ไม่ได้ผล

เมื่อเป็นนักบริหารระบบ

หลังจากที่ได้เป็น user ที่มีประสบการณ์ดีแล้ว ก็ถึงเวลาที่ควรจะได้รู้จักการเป็น system administrator ซึ่งก็โชคดีที่ไม่ได้เป็นสิ่งที่ยาก แต่อย่างไรก็ตามการเป็น s.a. ย่อมต้องมีสิทธิต่างๆมากกว่า user ธรรมดา super-user จะได้รับเครื่องหมาย prompt เป็นรูป # เพื่อให้เป็นการเตือนใจอยู่เสมอว่าเป็น super-user แล้วควรจะทำงานด้วยความระมัดระวัง

การเป็น super-user โดยการ login ในชื่อของ root โดยใช้รหัส Geheim หรือ login ด้วยชื่อของ su และ ใช้รหัสเดียวกัน จากนั้นท่านก็จะได้เป็น super-user

(1) สร้างระบบไฟล์

สิ่งหนึ่งที่ super-user ต้องทำก็คือการสร้างระบบไฟล์ (file system) ขึ้นมาใหม่ ซึ่งสามารถทำได้ 2 วิธี โดยวิธีแรก เมื่อเริ่ม boot ระบบ จะปรากฏเมนู ให้ผู้ใช้เลือก ซึ่งแต่เดิมถ้าผู้ใช้ต้องการเข้าสู่ระบบมินิกซ์ ก็ต้องกดคีย์เครื่องหมาย = แทน แต่สำหรับกรณีที่ต้องการสร้างระบบไฟล์ให้แผ่นดิสก์ใหม่ ก็ให้เลือกเมนูในหัวข้อสร้างระบบไฟล์ โดยกดคีย์อักษร

m จากนั้นเมื่อสร้างระบบไฟล์เป็นที่เรียบร้อยแล้วก็สามารถนำแผ่นดิสต์ไปใช้งานได้

สำหรับอีกวิธีหนึ่งเป็นการสร้างระบบไฟล์ในขณะที่ระบบ minix กำลังทำงานอยู่ นั่นคือเป็นการใช้คำสั่ง mkfs (make file system) โดยถ้าต้องการสร้างระบบไฟล์ขนาด 360 K ณ.ตำแหน่งไดรฟ์ B: ให้คีย์คำสั่งดังนี้

```
mkfs /dev/fd1 360
```

เมื่อโปรแกรมทำงานเป็นที่เรียบร้อยแล้ว แผ่นดิสต์ก็พร้อมที่จะนำไปใช้งาน สำหรับกรณีที่ระบบคอมพิวเตอร์มีเพียงไดรฟ์เดียว ให้ copy ไฟล์คำสั่ง mkfs ไปไว้ใน /bin จากนั้นให้ทำการ umount ไดรฟ์ A: โดยใช้คำสั่ง /etc/umount /dev/fd0 แล้วให้นำแผ่นดิสต์ที่จะสร้างระบบไฟล์ใส่ในไดรฟ์ A: แทน แล้วให้คีย์คำสั่งดังนี้

```
mkfs /dev/fd0
```

ก็เป็นอันว่าเป็นที่เรียบร้อยสำหรับการสร้างระบบไฟล์ลงในแผ่นดิสต์เพื่อใช้งาน

(2) การแก้ไขระบบไฟล์

ตามที่ได้อธิบายแล้วว่า ระบบ minix ประกอบด้วยระบบไฟล์ 3 ระบบ ซึ่งเก็บอยู่ในแผ่นดิสต์ คือ root file system , /usr file system , /user file system เมื่อระบบถูก boot root file system จะถูก copy ไปยังแรมดิสต์ แล้วแผ่นรูตดิสต์ (root file system) ก็จะไม่ถูกใช้อีกต่อไป ดังนั้น เมื่อเลิกใช้งานแล้ว การแก้ไขข้อมูลต่างๆ บนแรมดิสต์ ก็จะไม่ถูกเก็บลงในแผ่นดิสต์ แต่ต้องการเก็บข้อมูลของระบบไฟล์ที่แก้ไข ให้ทำการ mountแผ่นรูตดิสต์เข้าใน /user แล้วจากนั้นก็เป็นหน้าที่ของผู้ใช้เองที่จะต้องก็อปปี้ไฟล์ข้อมูลต่างๆที่จะเก็บลงในแผ่นดิสต์ให้เรียบร้อย

จะเห็นได้ว่า ระบบไฟล์ต่างๆทั้งสามที่กล่าวมา สามารถเปลี่ยนแปลงได้ แต่อย่างน้อยก็ควรมีเนื้อที่ว่างให้สำหรับ /tmp อย่างน้อย 50 K สรุปได้ว่า การแก้ไขระบบไฟล์ต่างๆในแผ่นดิสต์นอกจากจะทำกับระบบไฟล์ทั้งสามที่กล่าวมาแล้ว ยังสามารถจัดการกับแผ่นดิสต์อื่นๆ ได้ด้วย

ภาคผนวก ค.

การใช้ระบบปฏิบัติการมินิกซ์ภาษาไทยและโปรแกรมเชลล์

การใช้ระบบปฏิบัติการมินิกซ์ภาษาไทย

การใช้งานระบบปฏิบัติการมินิกซ์ภาษาไทยจะเหมือนกับการใช้ระบบปฏิบัติการมินิกซ์เดิมทุกประการ แต่มีความแตกต่างจากเวอร์ชัน 1.3 คือมีการแก้ไขและเพิ่มเติมฟังก์ชันคีย์ ดังนี้

1. ปุ่ม F3 (toggle software/hardware scrolling) ไม่มีใช้อีกต่อไป เพราะการแสดงผลระบบภาษาไทยมินิกซ์เป็นแบบกราฟฟิคโหมด ซึ่งไม่มีฮาร์ดแวร์สนับสนุนการสครอลล์หน้าจอ

2. เพิ่มปุ่ม F4 (show screen buffer contents) สำหรับการดีบัก โดยจะแสดงบัฟเฟอร์ที่เก็บตัวอักษรไทย/อังกฤษบนหน้าจอในขณะนั้น เพื่อใช้ทดสอบว่าบัฟเฟอร์เก็บถูกต้องหรือไม่ ปุ่มนี้จะไม่ได้ใช้ในการใช้งานปกติ

3. เพิ่มปุ่ม F8 (toggle screen mode) สำหรับเปลี่ยนโหมดการแสดงผลระหว่างระบบภาษาไทย/อังกฤษแบบไม่จัดระดับ 25 บรรทัด กับ ระบบภาษาไทย/อังกฤษจัดระดับตัวอักษร 17 บรรทัด

4. เพิ่มปุ่ม F9 (change font) ใช้ในการเปลี่ยนฟอนต์ (ลักษณะตัวอักษร) ในขณะใช้งาน โดยในระบบจะมีฟอนต์ติดตั้งอยู่แล้ว 2 รูปแบบ

5. เพิ่มปุ่ม F10 (toggle Thai/English key) เป็นปุ่มที่ใช้สำหรับกำหนดโหมดการป้อนตัวอักษรจากแป้นพิมพ์ว่าเป็นตัวอักษรภาษาไทยหรืออังกฤษ โดยสามารถทราบว่าขณะนั้นอยู่ในโหมดรับข้อความเป็นภาษาไทยหรืออังกฤษได้โดยการสังเกตที่รูปร่างของเคอร์เซอร์ซึ่งจะไม่เหมือนกัน

จากการกำหนดปุ่มเพิ่มเติมนั้น มีข้อสังเกตที่ว่า ปุ่ม F9 จะมีการกดฟังก์ชันคีย์เดียวกันกับปุ่ม Ctrl-F9 ซึ่งใช้ในการ kill ทุกโปรเซส ดังนั้นเมื่อมีการกดปุ่ม Ctrl-F9 แล้วนอกจากทุกโปรเซสจะถูกฆ่าแล้ว ยังเป็นการเปลี่ยนฟอนต์อีกด้วย

นอกจากนี้ปุ่มฟังก์ชันคีย์เหล่านี้ สามารถถูก disable ได้โดยการเรียกใช้ชิสเต็มคอลล์ เพื่อที่จะบังคับให้ผู้ใช้ไม่สามารถเปลี่ยนแปลงลักษณะการทำงานของโปรแกรม เช่น โปรแกรมที่ต้องใช้กับระบบภาษาไทย 17 บรรทัด สามารถสั่ง disable ปุ่ม F8 เพื่อมิให้ผู้ใช้เปลี่ยนโหมดกลับเป็นระบบภาษาไทย 25 บรรทัดเองได้ รายละเอียดโปรดดูได้จากภาคผนวกสรุปการใช้ชิสเต็มคอลล์ภาษาไทย

การใช้งานภาษาไทย สามารถใช้ได้เหมือนกับภาษาอังกฤษปกติ คือ สามารถตั้งชื่อไฟล์เป็นภาษาไทยได้ ตั้งชื่อยูสเซอร์เป็นภาษาไทยได้ แสดงผลภาษาไทยได้ทั้งแบบจัดระดับและไม่จัดระดับ แต่มีข้อจำกัดอยู่ที่ตัวอักษรภาษาไทยไม่สามารถใช้งานได้กับอิดิตเตอร์ mined และ elle แต่สามารถใช้งานได้กับอิดิเตอร์ ed ที่แก้ไขแล้ว

การใช้งานโปรแกรมเชลล์

โปรแกรมเชลล์เป็นโปรแกรมที่เพิ่มความสะดวกสบายในการใช้งานระบบปฏิบัติการมินิกซ์ให้ใช้ง่ายขึ้น โดยมีการทำงานเป็นแบบระบบเมนูซึ่งรวบรวมคำสั่งที่ใช้งานในระบบย่อยๆไว้เป็นหมวดหมู่และมีคำอธิบายประกอบการใช้งานตลอดเวลา โดยขั้นตอนการใช้งานโปแกรมเชลล์ให้ปฏิบัติตามดังนี้

1.ให้นำแผ่นดิสต์ที่เก็บโปรแกรม shell (shell.out) มาทำการ mount โดยคีย์คำสั่งดังนี้

```
/etc/mount /dev/fd1 /user
```

2.เปลี่ยน current directory ไปยัง /user โดยคีย์คำสั่งดังนี้

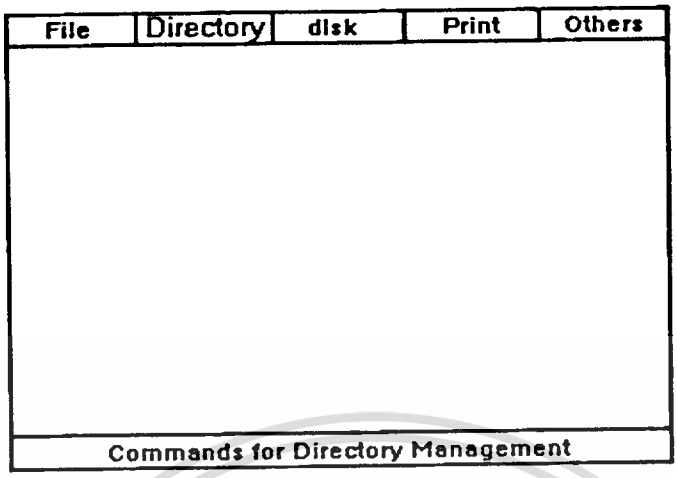
```
cd /user
```

3.ให้เปลี่ยนโหมดการเลื่อนจอภาพ ให้เป็น software scroll โดยกดคีย์ F3 แล้วตามด้วยกดคีย์ enter

4.จากนั้นให้โหลดโปรแกรม shell มาใช้งาน โดยคีย์คำสั่งดังนี้

```
shell.out
```

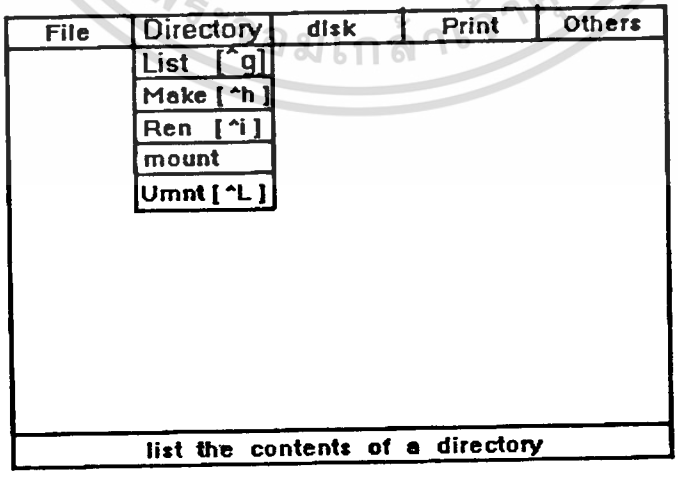
5.จะปรากฏเมนูขึ้นบนจอภาพซึ่งมีอยู่ 5 หัวข้อหลัก โดยมีแถบสว่างเป็นตัวเลือก ดังรูปค.1



รูป ค.1 แสดงเมนูหลักของเชลล์

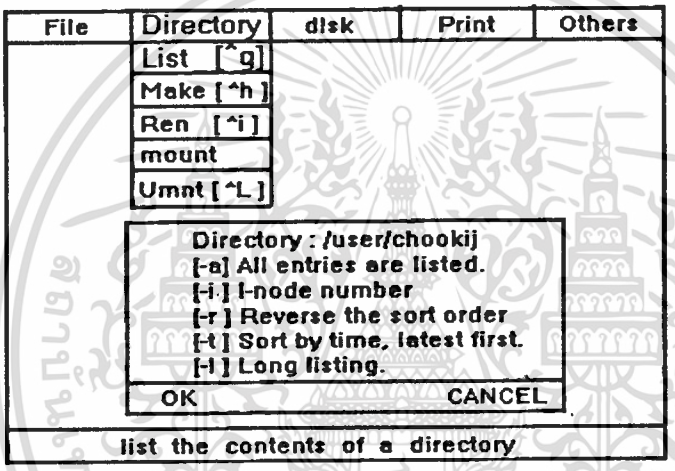
ให้ใช้คีย์ลูกศรขวา, คีย์ลูกศรซ้าย เลือกหัวข้อหลัก เมื่อเลือกหัวข้อหลักได้แล้วให้กดคีย์ enter

6.จากนั้นจะปรากฏเมนูย่อยของหัวข้อหลัก ซึ่งมีแถบสว่างเป็นตัวเลือกเช่นกัน โดยใช้คีย์ลูกศรขึ้น, คีย์ลูกศรลง เมื่อเลือกคำสั่งตามที่ต้องการได้แล้วให้กดคีย์ enter ซึ่งจะปรากฏ dialog box ให้ใส่ข้อมูลที่ต้องการ ดังรูปค.2



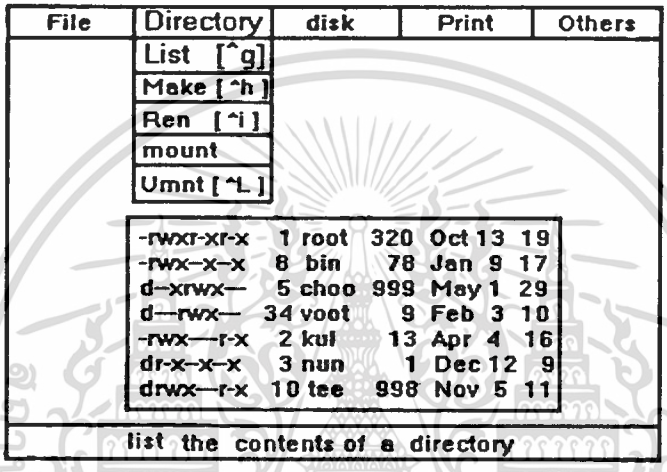
รูป ค.2 รูปแสดงรายละเอียดของเมนูย่อย

7.ภายใน dialog box จะมีลูกศร ชี้ตำแหน่งที่จะใส่ข้อมูล ซึ่งแบ่งออกเป็น 2 กลุ่ม คือ กลุ่มแรกให้ผู้ใช้ใส่ข้อมูลตามที่ต้องการ ส่วนกลุ่มที่สองการทำงานเป็นลักษณะ toggle โดยเมื่อกดคีย์ enter จะปรากฏรูปหัวใจ ซึ่งเป็นการบอกว่าต้องการหัวข้อนั้น แต่ถ้าไม่ต้องการให้กดคีย์ enter ซ้ำรูปหัวใจก็จะหายไป การทำงานก็จะเป็นในลักษณะนี้เรื่อยไป แสดงได้ดังรูปค.3 นี้



รูป ค.3 แสดงการใช้ dialog box

8.หลังจากที่ใส่ข้อมูลใน dielog box เรียบร้อยแล้ว โปรแกรมก็จะทำงานตามที่ผู้ใช้ต้องการ โดยส่วนผลลัพธ์จะมี window แสดงผลให้ทราบ ซึ่งในบางครั้งข้อความที่อยู่ในภายในอาจจะมีขนาดใหญ่กว่าขนาดของ window จึงต้องใช้คีย์ page-up, page-down, ลูกศรเลื่อนขึ้น, ลูกศรเลื่อนลง ในการดูข้อความในส่วนที่เกิน แสดงได้ดังรูปค.4



รูป ค.4 แสดงการแสดงผลลัพธ์ในหน้าต่าง

9.หลังจากใช้งานในส่วนแสดงผลเรียบร้อยแล้ว ให้กดคีย์ esc กลับสู่ระบบเมนูเพื่อเลือกการทำงานอื่นต่อไป แต่ถ้าต้องการเลิกใช้งานโปรแกรม ให้กดคีย์ esc กลับสู่เมนูหลัก แล้วให้กดคีย์ esc อีกครั้งหนึ่งก็จะกลับสู่ระบบมินิกซ์ (ปรากฏเครื่องหมายพร้อม $)

10. จะเห็นได้ว่าที่ระบบเมนู บางคำสั่งจะบอก hotkey ไว้ ซึ่งในบางครั้งผู้ใช้ต้องการความรวดเร็วในการใช้งาน ก็สามารถกด hotkey ได้เลย

ภาคผนวก ง.

การพัฒนาโปรแกรมบนมินิกซ์และเอ็มเอสดอส

การพัฒนาโปรแกรมบนมินิกซ์

มินิกซ์สามารถใช้งานได้ตั้งแต่คอมพิวเตอร์ 16 บิต (IBM PC/XT) หน่วยความจำ 640K และฟลอปปี้ดิสค์ 2 ตัว แต่ในการพัฒนาโปรแกรมควรจะมีระบบที่ใหญ่กว่านี้ เพื่อความสะดวกรวดเร็ว เช่น ควรมีการติดตั้งรู้ตดิสค์ (รู้ตดิสค์เป็นแผ่นดิสค์ที่เก็บรู้ตไฟล์ซิสเต็มตั้งแต่รากลงมาและจะต้องเก็บไว้ในแรมดิสค์เสมอ โดยระบบจะจองขนาดแรมดิสค์โดยกันเนื้อที่ในหน่วยความจำหลักไว้เท่ากันกับขนาดของรู้ตดิสค์) ไว้ในหน่วยความจำขยาย (extended memory) ซึ่งทำได้โดยการสร้างแผ่นรู้ตดิสค์ที่มีขนาดเกิน 256K ระบบจะก็อปปี้รู้ตดิสค์ลงไปในหน่วยความจำขยาย(ถ้ามี)โดยอัตโนมัติ

ระบบที่เหมาะสมสำหรับการพัฒนา คือ ระบบที่ใช้เครื่องคอมพิวเตอร์ 32 บิต (80386) ซึ่งจะมีความเร็วในการทำงานสูง มีหน่วยความจำ 2 เมกกะไบต์ใช้สำหรับเก็บรู้ตไฟล์ซิสเต็มขนาด 544K (ใช้เก็บคอมมานด์โปรแกรมที่จำเป็นรวมทั้งคอมไพเลอร์และไลบรารีด้วย) และมีฟลอปปี้ดิสค์ขนาด 1.2M อย่างน้อย 1 ตัว เพื่อใช้เก็บโปรแกรมที่จะพัฒนา แต่ถ้ามีเพียงระบบที่ใช้เครื่องคอมพิวเตอร์ 16 บิต หน่วยความจำ 640K ควรจะมีฟลอปปี้ดิสค์ 2 ตัว โดยไดร์ฟแรกเก็บคอมมานด์โปรแกรมที่จำเป็นจริงๆรวมทั้งคอมไพเลอร์และไลบรารี และไดร์ฟที่เหลือใช้เก็บโปรแกรม

การแก้ไขโปรแกรมที่มีข้อความภาษาไทย

เนื่องจากอิดิตเตอร์มินิอิดิตและแอลเล่ไม่สนับสนุนการแก้ไขไฟล์ที่มีตัวอักษรภาษาไทย ดังนั้นการแก้ไขโปรแกรมหรือเอกสารที่มีข้อความภาษาไทย สามารถทำได้ด้วยวิธีดังต่อไปนี้

1. เนื่องจากเชลล์ (sh) สามารถรับข้อความภาษาไทยได้ ดังนั้นเราสามารถใช้เชลล์ร่วมกันกับการเปลี่ยนทิศทางไฟล์ (redirect) และคำสั่ง echo สำหรับการเขียนข้อความสั้นๆลงในไฟล์ เช่น

echo นี่คือเอกสารทดสอบการป้อนข้อความภาษาไทย >เอกสารทดลอง

echo บรรทัดนี้เป็นบรรทัดที่สอง >>เอกสารทดลอง

เป็นการป้อนข้อความขนาดสองบรรทัดลงในไฟล์ชื่อ "เอกสารทดลอง" โดยเมื่อ cat ไฟล์นี้ จะได้ผลลัพธ์ดังนี้

```
# cat ไฟล์ทดลอง
นี่คือเอกสารทดสอบการป้อนข้อความภาษาไทย
บรรทัดนี้เป็นบรรทัดที่สอง
```

2. ใช้อิดิตเตอร์ ed แทน โดย ed เป็นอิดิตเตอร์ที่สามารถแก้ไขเอกสารได้ทีละหนึ่งบรรทัด โดยได้มีการแก้ไขให้สามารถรับและแสดงผลข้อความภาษาไทยได้ และมีการใช้งานเหมือนกันกับ ed บนยูนิกซ์

3. เขียนโปรแกรมสำหรับพัฒนาบนเอ็มเอสดอส โดยใช้อิดิเตอร์บนดอสที่มีความสามารถสูงได้ตามความต้องการ และเมื่อจะใช้งานก็สามารถถ่ายไปยังมินิกซ์ได้โดยการใช้คำสั่ง dosread เช่น เมื่อเขียนโปรแกรม test.c บนเอ็มเอสดอส สามารถถ่ายไปยังมินิกซ์ด้วยการใช้คำสั่งบนมินิกซ์ ดังนี้

```
dosread 1 -a test.c >test.c
```

เป็นการอ่านไฟล์ test.c บนดอสจากไดร์ฟหมายเลข 1 (ไดร์ฟ B) และเอาท์พุตถูกเปลี่ยนทิศทางไปออกยังไฟล์ที่ชื่อ test.c บนมินิกซ์ สำหรับตัวเลือก -a หมายถึงไฟล์ test.c เป็นเท็กซ์ไฟล์ที่เก็บรหัสแอสกี้ นอกจากโปรแกรม dosread แล้วยังมีโปรแกรมอื่นๆที่เกี่ยวข้องคือ โปรแกรม dosdir สำหรับแสดงไฟล์ในไดเรกทอรีของแผ่นดอส และโปรแกรม doswrite สำหรับเขียนไฟล์จากมินิกซ์ลงไปเก็บในแผ่นดอส สำหรับรายละเอียดเพิ่มเติมสามารถหาได้จากภาคผนวก ฉ.

สำหรับการดูไฟล์เอกสารภาษาไทยสามารถทำได้โดยการใช้คำสั่ง cat หรืออาจจะใช้โปรแกรม browser ซึ่งเป็นโปรแกรมแสดงเอกสารเป็นระบบหน้าต่าง โดยสามารถดูรายละเอียดการใช้งานได้ในบทที่ 8

การกำหนดสภาพแวดล้อมการทำงานบนมินิกซ์

เนื่องจากมินิกซ์มีระบบไฟล์รากอยู่ที่แรมดิสค์ การติดตั้งระบบไฟล์เพิ่มจะต้องทำการเมานท์ระบบไฟล์จากดิไวซ์ฟลอปปี้ดิสค์เสมอ เมื่อมีการสลับแผ่นเปลี่ยนเข้า/ออก ก็จะต้องทำการเมานท์ดิไวซ์ฟลอปปี้ดิสค์ออกก่อนเสมอ โดยการใช้คอมมานด์โปรแกรม mount และ umount เนื่องจากพารามิเตอร์ที่ใช้ในการเมานท์ระบบไฟล์มีความยาวพอสมควร อีกทั้งยังมีความบ่อยในการเรียกใช้ ทำให้เสียเวลาในการป้อนคำสั่งเรียกใช้มาก จึงได้กำหนดตัวแปรเอนไวรอนเมนต์สำหรับทำการเมานท์แผ่นเข้า/ออก สำหรับไดรฟ์ 1.2M (ชื่อดีไวซ์ /dev/at0 สำหรับไดรฟ์ A และ /dev/at1 สำหรับไดรฟ์ B) และไดรฟ์ 360K (ชื่อดีไวซ์ /dev/fd0 สำหรับไดรฟ์ A และ /dev/fd1 สำหรับไดร์ฟ B) ดังนี้

```
mat0='/etc/mount /dev/at0 /usr'
mfd0='/etc/mount /dev/fd0 /usr'
mfd1='/etc/mount /dev/fd1 /user'
umat0='/etc/umount /dev/at0'
umfd0='/etc/umount /dev/fd0'
umfd1='/etc/umount /dev/fd1'
export mat0 mfd0 mfd1 umat0 umfd0 umfd1
```

ซึ่งชุดคำสั่งเหล่านี้สามารถป้อนทีละบรรทัดก็ได้ หรืออาจจะเก็บไว้ในไฟล์ที่ชื่อว่า (/.profile) เป็นต้น เมื่อมีการล็อกอินเข้ามาในระบบแล้ว ไฟล์นี้จะถูกสั่งทำงานโดยอัตโนมัติ

การเรียกใช้ตัวแปรเอนไวรอนเมนต์ทำได้โดย การป้อนเครื่องหมายดอลล่าร์ ($) ก่อนชื่อตัวแปรเอนไวรอนเมนต์ เช่น

```
# $mat0
```

จะทำให้เชลล์ไปสั่งให้คำสั่ง /etc/mount /dev/at0 ทำงาน

ภาคผนวก จ.

เทคนิคการใช้คอมไพเลอร์และแอสเซมเบลอร์

การพัฒนาระบบปฏิบัติการมินิกซ์เพื่อให้มีความสามารถเพิ่มขึ้น จำเป็นจะต้องมีการเขียนโปรแกรมทั้งภาษาสูงและภาษาแอสเซมบลี้ ดังนั้นคอมไพเลอร์และแอสเซมเบลอร์จึงเป็นสิ่งที่ขาดเสียไม่ได้ สำหรับมินิกซ์มีซอฟท์แวร์คอมไพเลอร์ภาษาซีและแอสเซมเบลอร์ของตัวเอง ซึ่งมีรูปแบบแตกต่างจากซอฟท์แวร์บน MS-DOS อยู่บ้าง ซึ่งจะกล่าวถึงต่อไป ก่อนที่จะนำตัวแปลภาษาเหล่านี้ไปใช้งาน ควรจะทราบการทำงานอย่างคร่าวๆของตัวแปรภาษาเหล่านี้ ตลอดจนเทคนิคการใช้ตัวแปรภาษาบนมินิกซ์ นอกเหนือจากที่ได้กล่าวไว้ในภาคผนวกการใช้งานคอมมานด์โปรแกรม

คอมไพเลอร์ cc

คอมไพเลอร์ภาษาซีบนมินิกซ์ จะประกอบไปด้วยโปรแกรมหลักชื่อว่า cc ที่เขียนด้วยภาษาซีเอง โดยมีหน้าที่ในการรับพารามิเตอร์ที่จะใช้ในการคอมไพล์โปรแกรม และส่งต่อไปยังส่วนอื่นๆ ทั้งสิ้น 5 ส่วนเรียงตามลำดับ โดยเอาท์พุทจากส่วนหนึ่งจะกลายเป็นอินพุทของส่วนถัดไป เช่น ในการคอมไพล์โปรแกรม test.c โดยเรียกใช้คำสั่ง cc test.c จะทำให้เกิดการเรียกใช้โปรแกรมทั้ง 5 ส่วนทำงานตามลำดับดังนี้

1. โปรแกรม cpp จะทำการพรีโปรเซสโปรแกรมภาษาซี เช่น การทำมาโครในคำสั่ง #define, การอ่านไฟล์เพิ่มจากคำสั่ง #include เป็นต้น

test.c -----> test.i

2. โปรแกรม cem เป็นส่วนที่จะกระจายโปรแกรมภาษาซีออกเป็นหน่วยย่อยและทำการวิเคราะห์โครงสร้างของโปรแกรม และสร้างโค้ดที่มีรูปแบบเฉพาะขึ้น

test.i -----> test.k

3. โปรแกรม opt เป็นโปรแกรมที่ทำวิเคราะห์โค้ดเพื่อที่จะลดขนาดของโค้ดเพื่อเพิ่มประสิทธิภาพของโปรแกรมผลลัพธ์ ซึ่งเรียกว่าเป็นการทำ optimization

test.k -----> test.m

4. โปรแกรม cg เป็นส่วน code generation หรือจะทำการสร้างโค้ดออกมาเป็นไฟล์ภาษาแอสเซมบลี้ เพื่อใช้ในส่วนถัดไป

test.m -----> test.s

5. โปรแกรม asld จะทำการแอสเซมเบลอร์ไฟล์ที่ได้จากส่วนก่อน และนำมาลิงค์เข้ากับไลบรารีภาษาซี เพื่อที่จะสร้างไฟล์ที่สามารถ execute ได้

test.s -----> a.out

<u>เทคนิคการใช้คอมไพเลอร์ cc</u>

เนื่องจาก cc เป็นคอมไพเลอร์แบบ 5 ส่วน (five-pass compiler) โดยแบ่งการทำงานออกเป็น 5 โปรเซส โดยแต่ละโปรเซสติดต่อกันผ่านไฟล์อินพุทและเอาท์พุทเท่านั้น คอมไพล์เลอร์ cc จึงเรียกใช้โปรเซส cpp และ cem พร้อมๆกันโดยการทำ fork และทั้งสองโปรเซสจะติดต่อกันผ่าน pipe ซึ่งจะทำให้การคอมไพล์โปรแกรมได้เร็วขึ้น เนื่องจากส่วนเอาท์พุทของ cpp จะถูกเปลี่ยนทาง (redirect) ไปยังส่วนอินพุทของ cem โดยผ่านทางไปป์ (stdout, stdin) โดยที่ไม่ต้องเขียนเอาท์พุทออกมาเป็นไฟล์ แต่วิธีนี้ไม่เหมาะสำหรับระบบที่มีหน่วยความจำหลักน้อย เช่น เครื่องที่มีหน่วยความจำเพียง 512K หรือ ระบบที่ติดตั้งแรมดิสค์ขนาดใหญ่ทำให้หน่วยความจำเหลือให้ใช้มีน้อยลง เป็นต้น ซึ่งถ้าหน่วยความจำหลักมีขนาดไม่พอ ก็จะไม่สามารถคอมไพล์โปรแกรมได้

เราสามารถสั่งให้คอมไพเลอร์ cc เรียกใช้ cpp และ cem ทำงานแยกกันทีละโปรเซส โดยใช้ออปชัน -F ซึ่งจะบังคับให้ทั้งสองโปรแกรมติดต่อกันโดยผ่านไฟล์แทน มีผลให้การคอมไพล์โปรแกรมใช้เวลานานขึ้น และต้องใช้เนื้อที่ดิสค์สำหรับเก็บไฟล์เอาท์พุทของ cem

ไฟล์ชั่วคราวต่างๆที่ถูกสร้างขึ้นในระหว่างการคอมไพล์โปรแกรม จะถูกจัดเก็บไว้ในพาธ /tmp ซึ่งเมาท์ (mount) อยู่ในดิไวซ์แรมดิสค์ที่มีความเร็วสูง แต่จะมีปัญหาเกิดขึ้นถ้าไฟล์ที่นำมาคอมไพล์มีขนาดใหญ่หรือในขณะลิงค์ไฟล์จำนวนมาก และแรมดิสค์มีขนาดไม่พอที่จะเก็บไฟล์เหล่านี้ทั้งหมด จะทำให้เกิดข้อผิดพลาด no space on root device ขึ้น

ทางแก้ก็คือการเปลี่ยนพาธที่ใช้เก็บไฟล์ชั่วคราวไว้ในดิไวซ์ฟลอปปี้ดิสค์แทน โดยถ้าไดเรกทอรีปัจจุบันอยู่ในดิไวซ์ฟลอปปี้ดิสค์ เช่นพาธ /usr/kernel และในฟลอปปี้ดิสค์มีเนื้อที่ว่าง

เพียงพอ ก็สามารถสั่งให้คอมไพเลอร์ใช้ไดเรกทอรีปัจจุบันเก็บไฟล์ชั่วคราว โดยใช้ออปชัน -T. ซึ่งเครื่องหมาย . หมายถึงไดเรกทอรีปัจจุบันนั่นเอง

ตัวคอมไพเลอร์ cc เอง มีซอร์สโค้ดอยู่ในแผ่นคอมมานด์ดิสค์ชื่อว่าไฟล์ cc.c ซึ่งสามารถคอมไพล์เป็นไฟล์ cc ได้ โดยการเรียกใช้ cc -o cc -DMEM640K cc.c สำหรับระบบที่มีหน่วยความจำ 640K และใช้ออปชัน -DMEM512K แทนสำหรับระบบที่มีหน่วยความจำขนาด 512K หรือใช้ออปชัน -DRAMDISK สำหรับระบบที่มีขนาดแรมดิสค์ใหญ่มากๆ (มักเป็นระบบที่มีหน่วยความจำขยาย หรือ extended memory และติดตั้งขนาดแรมดิสค์เกิน 256K - อ่านรายละเอียดได้ในหัวข้อการติดตั้งแผ่นรีดดิสค์) ซึ่งแต่ละออปชันมีข้อแตกต่างกันตรงที่พาธที่ใช้เก็บคอมไพล์เลอร์ส่วนต่างๆและไลบรารีจะอยู่ในพาธที่แตกต่างกัน เช่น เก็บอยู่ในพาธ /usr/lib ซึ่งอยู่ในฟลอปปี้ดิสค์ หรือเก็บอยู่ในพาธ /lib ซึ่งอยู่ในแรมดิสค์ เป็นต้น

การเก็บคอมไพเลอร์ทั้ง 5 ส่วน และ/หรือไลบรารีต่างๆ ไว้ในแรมดิสค์จะทำให้การคอมไพล์โปรแกรมเร็วขึ้นมากๆ แต่ก็จะต้องใช้เนื้อที่ในแรมดิสค์ขนาดใหญ่ด้วยเช่นกัน จึงไม่เหมาะสำหรับระบบที่มีหน่วยความจำ 640K แต่ไม่มีหน่วยความจำขยาย (สำหรับแผ่นดิสค์มินิกซ์สำหรับพัฒนาภาษาไทยถูกติดตั้งให้ใช้กับระบบที่มีหน่วยความจำอย่างต่ำ 2 เมกกะไบต์ และใช้ฟลอปปี้ดิสค์ขนาด 1.2M อย่างน้อยหนึ่งตัว ได้กำหนดขนาดแรมดิสค์ขนาด 544K และเก็บคอมไพเลอร์และไลบรารีทั้งหมดไว้ในแรมดิสค์)

ถ้าต้องการคอมไพล์โปรแกรมอย่างเดียวโดยไม่ต้องการลิงค์ ให้ใช้ออปชัน -c หรือ -S ก็ได้ คอมไพเลอร์ cc ก็จะข้ามขั้นตอนที่ห้าไป และโปรแกรมจะได้เอาท์พุทเป็น test.s แทน ซึ่งสามารถนำไปลิงค์กับโปรแกรมอื่นๆและไลบรารีภาษาซีในภายหลังได้ เช่นเรียก cc test.s คอมไพเลอร์จะไปเรียกใช้โปรแกรม asld ทันที โดยเรียกใช้พร้อมกับพารามิเตอร์ดังนี้

```
asld /usr/lib/crtso.s test.s /usr/lib/libc.a /usr/lib/end.s
```

โดยไฟล์ crtso.s จะเก็บส่วน startup code ซึ่งเป็นโค้ดแรกที่จะทำงานหลังจากการ exec โดยโค้ดส่วนนี้จะเริ่มต้นที่แอดเดรสศูนย์ (virtual address 0) มีหน้าที่ในการรับพารามิเตอร์มาจัดเก็บไว้ในตัวแปร argc, argv เซตตำแหน่งแสต็ก และเป็นเรียกใช้ฟังก์ชัน main อีกทอดหนึ่ง สำหรับไฟล์ libc.a จะเก็บฟังก์ชันมาตรฐานต่างๆเป็นไลบรารีไว้ เช่น ฟังก์ชัน fork, getc, printf เป็นต้น สำหรับไฟล์ end.s จะเก็บสัญลักษณ์บอกจุดสุดท้ายของ

โปรแกรมเพื่อใช้ในการลิงค์โปรแกรม

เราสามารถตรวจดูการเรียกใช้คอมไพเลอร์ส่วนต่างๆได้โดยการใช้ออปชัน -v เช่น cc -v test.c ซึ่งจะแสดงการเรียกใช้คอมไพเลอร์ทีละส่วนพร้อมกับพารามิเตอร์ที่ส่งให้แต่ละส่วนอย่างชัดเจน เมื่อพบปัญหาในระหว่างการคอมไพล์ ก็สามารถหาทางแก้ได้เช่น เพิ่มการใช้ออปชัน -F หรือ -T ให้เหมาะสม สำหรับออปชันอื่นมีรายละเอียดกล่าวไว้ในภาคผนวก

ข้อจำกัดบางประการของคอมไพเลอร์ cc

คอมไพเลอร์ cc บนมินิกซ์ เป็นคอมไพเลอร์ขนาดเล็กและมีข้อจำกัดพอสมควร ทำให้เกิดความยากลำบากในการ port โปรแกรมมายังมินิกซ์ โดยข้อจำกัดบางประการที่พบมีดังนี้

1. ทำ bit-field structure ไม่ได้ เช่น เราไม่สามารถกำหนด

```
struct BitField {
    unsigned SaraUp : 4;
    unsigned SaraDown : 2;
    unsigned Vannayuk : 4;
};
```

จะคอมไพล์ไม่ผ่าน โดยคอมไพเลอร์จะบอกว่าไม่สนับสนุนการทำ bit field ทางแก้ก็คือจัดการกับข้อมูลเป็นบิตโดยผ่านทางฟังก์ชันแทน ซึ่งเป็นเทคนิคที่ใช้ในการจัดเก็บบัฟเฟอร์ภาษาไทยในมินิกซ์

2. กำหนด nested comment ไม่ได้ เช่น เราไม่สามารถกำหนดคอมเมนต์คร่อมฟังก์ชันที่มีการคอมเมนต์ไว้อยู่แล้วได้ ดังตัวอย่าง

```
/*
/* stub function */
test()
{                       /* no routine at all */
```

```
}
*/
```

เราสามารถหาทางแก้ได้โดยการใช้ #ifdef, #endif เข้าช่วยดังนี้

```
#ifdef STRIP_CODE
/* stub function */
test()
{                       /* no routine at all */
}
#endif
```

3. กำหนด function prototype ตามแบบมาตรฐาน ANSI C แบบใหม่ไม่ได้ คือ เราไม่สามารถกำหนด

```
long filelength(char *);
main() {
    long size = filelength("/tmp/dummy.fil");
    ...
}
```

เราไม่สามารถบอกชนิดของพารามิเตอร์ต่างๆและจำนวนพารามิเตอร์ในโปรโตไทป์ได้ จะบอกได้เพียงแต่ชนิดของข้อมูลที่ฟังก์ชันรีเทิร์นกลับมาเท่านั้น ดังนี้

```
long filelength();
main()
{
    long size = filelength("/tmp/dummy.fil");
    ...
}
```

4. คำสั่งภาษาซีในบรรทัดสุดท้ายจะต้องจบด้วยรหัสขึ้นบรรทัดใหม่เท่านั้น นั่นคือบรรทัดสุดท้ายของโปรแกรมจริงๆควรจะเป็นบรรทัดว่างๆหลังจากบรรทัดที่เก็บคำสั่งภาษาซีคำสั่งสุดท้าย

มิฉะนั้นคอมไพเลอร์จะไม่คอมไพล์คำสั่งบรรทัดสุดท้ายทำให้เกิดข้อผิดพลาดขึ้น

5. ชื่อตัวแปร, ชื่อฟังก์ชัน จะมีความหมายเพียง 7 ตัวแรกเท่านั้น (ไม่รวมเครื่องหมายขีดเส้นใต้หน้าชื่อซึ่งคอมไพเลอร์เป็นผู้ใส่เพิ่มให้) ดังนั้นตัวแปร next_move กับ next_motion จะถือว่าเป็นตัวแปรเดียวกัน

แอสเซมเบลอร์ asld

แอสเซมเบลอร์บนมินิกซ์มีชื่อเหมือนกับแอสเซมเบลอร์บนยูนิกซ์ นั่นคือ asld (assembler and loader) จะทำการแอสเซมเบลอร์ไฟล์ภาษาแอสเซมบลี้ (ไฟล์ที่มีนามสกุลเป็น .s) และลิงค์โปรแกรมทั้งหมดออกเป็นไฟล์ที่สามารถ execute ได้ ออปชันต่างๆที่ใช้ใน asld สามารถดูรายละเอียดได้จากภาคผนวก

โปรแกรม asld บนมินิกซ์ จะใช้กับแอสเซมบลี้ของเครื่องไอบีเอ็มพีซี โดยมีข้อแตกต่างจากแอสเซมเบลอร์บน MS-DOS อยู่บ้าง พอสังเกตได้ดังนี้

1. คอมเมนต์จะใช้สัญลักษณ์ ';' แทน ';'

2. ตัวเลขเขียนแบบภาษาซี คือ ถ้าตัวอักษรแรกอยู่ระหว่างศูนย์ถึงเก้าหมายถึงเลขฐานสิบ เช่น 15, -8, 40000 เป็นต้น ถ้าตัวอักษรแรกคือศูนย์หมายถึงเลขฐานแปด เช่น 007, 010, 0377 เป็นต้น ถ้าสองตัวอักษรแรกคือ 0x หมายถึงเลขฐานสิบหก เช่น 0x0F, 0xFFFF เป็นต้น

3. ค่าคงที่ที่เป็นตัวเลข จะต้องใช้เครื่องหมาย '#' หรือเครื่องหมาย '*' นำหน้า เช่น mov ax,*0xFF ไม่ใช่ mov ax,0xFF หรือ shr ax,#1 ไม่ใช่ shr ax,1

4. คำสั่งที่เกี่ยวข้องกับการประมวลผลกับข้อมูลเป็นไบต์ จะลงท้ายด้วยตัวอักษร b เสมอ เช่น movb al,ch ไม่ใช่ mov al,ch

5. การอ้างข้อมูลแบบ indirect จะใช้เครื่องหมายวงเล็บแทนเครื่องหมายก้ามปู และค่าคงที่ที่อ้างจะต้องเขียนอยู่ข้างหน้าวงเล็บ เช่น mov ax,*2(bp) ไม่ใช่ mov ax,[bp+2]

6. ใช้คำสั่ง stow, stob, movw, movb แทน stosw, stosb, movsw, movsb

7. การกำหนด constant ให้ใช้เครื่องหมาย '=' แทนการใช้ EQU

8. การเขียนฟังก์ชันที่เรียกใช้ในโปรแกรมภาษาซี จะต้องเขียนฟังก์ชันนั้นขึ้นต้นด้วยเครื่องหมายขีดเส้นใต้ เช่น _Gputchar เป็นต้น

ภาคผนวก ฉ.

คำสั่งคอมมานด์ในมินิกซ์

คำสั่ง : asld (assembler-loader)

รูปแบบ : asld [-d] [-s] [-o name] file ...

ตัวเลือก : -L หมายถึง ให้แสดงลิสติ้งบนเอาท์พุทมาตรฐาน (standard output)

-T หมายถึง ให้ใช้ไดเรกทอรีสำหรับสร้างไฟล์ชั่วคราว

-o หมายถึง ผลลัพธ์ออกมาในอยู่ในไฟล์ที่ถูกตั้งชื่อโดยพารามิเตอร์ถัดไป

-s หมายถึง ให้แสดงตารางของสัญญาลักษณ์หรือตัวแปรต่างๆบนเอาท์พุท มาตรฐาน

หน้าที่ : เป็นตัวคอมไพล์ภาษาแอสเซมบลี้และเป็นตัวโหลดเดอร์

ตัวอย่าง : asld -s file.s

หมายถึง ให้แอสเซมบลี้ file.s และ แสดงสัญญาลักษณ์

asld -o output file.s

หมายถึง ให้แอสเซมบลี้ file.s โดยให้ผลลัพธ์อยู่ในไฟล์ output

asld -T. file1.s file2.s

หมายถึง ให้แอสเซมบลี้ file1.s และ file2.s โดยใช้ไดเรกทอรีขณะนั้นสร้างไฟล์ชั่วคราว

คำสั่ง : cat (concatenate files and write to standard output)

รูปแบบ : cat [-u] file ...

ตัวเลือก : -u หมายถึง ไม่ต้องใช้บัฟเฟอร์สำหรับผลลัพธ์ที่ออกมา

หน้าที่ : ใช้ในการต่อไฟล์หรือแสดงไฟล์หลายๆไฟล์ออกทางเอาท์พุทมาตรฐาน

ตัวอย่าง : cat file

หมายถึง แสดงผลรายละเอียดของไฟล์ (displayed file) ออกจอภาพ

cat file1 file2 | lpr

หมายถึง ให้ทำการต่อไฟล์ 2 ไฟล์ และให้ผลลัพธ์พิมพ์ออกทางเครื่องพิมพ์

ค่าสั่ง : cc (C compiler)

รูปแบบ : cc [option] ... file ...

ตัวเลือก : -Dx=y หมายถึง ใช้ในการนิยามมาโคร x ให้มีค่าเท่ากับ y

-F หมายถึง ใช้ไฟล์เก็บเอาท์พุทจากส่วนพริโปรเซสเซอร์แทนการใช้ไปป์

-Idir หมายถึง ใช้บอกไดเรกทอรีที่เก็บไฟล์ include

-LIB หมายถึง สร้างโมดูลไลบรารี

-R หมายถึง บอกคำเตือนถึงโค้ดที่ไม่กำหนดตาม Kernighan & Ritchie

-S หมายถึง คอมไพล์อย่างเดียวได้เอาท์พุทเป็นไฟล์ .s แล้วหยุด

-Tdir หมายถึง บอกไดเรกทอรี สำหรับให้ cem ใช้เก็บไฟล์ชั่วคราว

-U หมายถึง เลิกนิยามมาโคร

-c หมายถึง เหมือนกับ -S

-o หมายถึง เก็บชื่อเอาท์พุทตามชื่อไฟล์ที่กำหนดหลังแฟลกนี้

-v บอกขั้นตอนในการคอมไพล์

-w ไม่พิมพ์ข้อความเตือน

หน้าที่ : เป็นคอมไพเลอร์ภาษาซี

ตัวอย่าง :

```
cc -c file.c              # คอมไพล์โปรแกรม file.c อย่างเดียว
cc -Di8088 file.c         # นิยามสัญญลักษณ์ i8088
cc -c -LIB file.c         # สร้างเป็นโมดูลสำหรับทำไลบรารี
cc -R -o out file.c       # ตรวจเช็ค K & R และเอาท์พุทไปที่ไฟล์ out
cc -T. -F file.c          # ใช้ไดเรกทอรีปัจจุบัน และใช้ไฟล์เก็บแทนไปป์
```

โปรแกรมคอมไพเลอร์ภาษาซี เป็นแบบมัลติพาส (multipass) โดยประกอบไปด้วย 5 โปรเซสดังนี้

โปรแกรม	อินพุท	เอาท์พุท	งานที่ทำ

/lib/cpp	prog.c	prog.i	ทำพรีโปรเซสเซอร์ เช่น #include,#define
/lib/cem	prog.i	prog.k	กระจายประโยคและวิเคราะห์โครงสร้าง
/usr/lib/opt	prog.k	prog.m	ทำการลดขนาดของโค้ดให้มีประสิทธิภาพ
/usr/lib/cg	prog.m	prog.s	สร้างโค้ด
/usr/lib/asld	prog.s	a.out	ทำการแอสเซมเบลอร์และลิงค์

โปรแกรม cc จะทำการฟอร์คโปรเซสไปเรียกโปรแกรมส่วนอื่นๆอย่างเหมาะสม และผ่านแฟลกและพารามิเตอร์ไปยังโปรแกรมนั้นๆด้วย โดยแฟลก -v จะแสดงการเรียกใช้แต่ละส่วน สำหรับแฟลก -c หรือ -S

คำสั่ง : chmod (change file mode)

รูปแบบ : chmod mod file ...

ตัวเลือก : ไม่มี

หน้าที่ : ใช้สำหรับเปลี่ยนลักษณะของไฟล์ เช่น การอ่าน การเขียน หรือ การเอ็กซิคิว

ตัวอย่าง : chmod 754 file

หมายถึง ให้เปลี่ยนลักษณะไฟล์ดังนี้ Owner: rwx; Group r-w; Other r--

chmod 4777 file1 file2

หมายถึง ให้เซต SETUID บิทด้วย (ถ้าหากเปลี่ยนจาก 4 เป็น 2 แล้วจะหมายถึงเซต SETGID)

คำสั่ง : chown (change owner)

รูปแบบ : chown user file ...

ตัวเลือก : ไม่มี

หน้าที่ : ใช้สำหรับเปลี่ยนชื่อเจ้าของไฟล์จากคนหนึ่งไปยังอีกคนหนึ่ง คำสั่งนี้ใช้ได้เฉพาะ super-user เท่านั้น

ตัวอย่าง : chown ast file1 file2

หมายถึง เปลี่ยนเจ้าของไฟล์ file1 และ file2 เป็น ast

คำสั่ง : clr (clear the screen)

รูปแบบ : clr

ตัวเลือก : ไม่มี

หน้าที่ : ลบจอภาพ

ตัวอย่าง : clr

คำสั่ง : cp (copy file)

รูปแบบ : cp file1 file2

cp file ... directory

ตัวเลือก : ไม่มี

หน้าที่ : ก๊อปปี้ไฟล์จาก file1 ไป file2 หรือไปยังไดเรกทอรี

ตัวอย่าง : cp oldfile newfile

หมายถึง ก๊อปปี้จากไฟล์ oldfile เป็น newfile

cp file1 file2 /tmp

หมายถึง ก๊อปปี้ไฟล์ file1 และ file2 ลงในไดเรกทอรี tmp

คำสั่ง : date (print or set the date and time)

รูปแบบ : date [[MMDDYY]hhmm[ss]]

ตัวเลือก : -q หมายถึง ให้ทำการตั้งเวลาโดยอ่านข้อมูลจากคีย์บอร์ด

หน้าที่ : สำหรับบอกหรือตั้งเวลาให้กับระบบ

ตัวอย่าง : date

หมายถึง พิมพ์วันและเวลาออกจอภาพ

date 0408911010

หมายถึง ตั้งเวลาให้เป็น วันที่ 8 เดือนเมษายน 1997 เวลา 10:10

คำสั่ง : df (disk space and i-nodes)

รูปแบบ : df special ...

ตัวเลือก : ไม่มี

หน้าที่ : ใช้สำหรับตรวจสอบเนื้อที่และไอโหนดบนดิสค์

ตัวอย่าง : df /dev/ram

หมายถึง ให้ตรวจสอบเนื้อที่ของแรมดิสค์

df~

หมายถึง ให้ตรวจสอบเนื้อที่ของแรมดิสค์และไดร์ฟที่ถูกเมานท์เข้าไป

คำสั่ง : dosdir (list and MS-DOS diskette directory)

รูปแบบ : dosdir [-lr] drive

ตัวเลือก : -l หมายถึง ให้แสดงผลทั้งหมดคือแสดงวันที่ ชื่อไฟล์ (แบบยาว)

-r หมายถึง ถ้าหากมีไดเรกทอรีก็ให้แสดงไฟล์ในไดเรกทอรีด้วย

หน้าที่ : ใช้สำหรับไดเรกทอรีที่เป็นแผ่นดอส

ตัวอย่าง : dosdir -l 1

หมายถึง ให้แสดงไฟล์ในแผ่นดอสในไดร์ฟ 1 แบบยาว

คำสั่ง : dosread (read a file from and MS-DOS diskette)

รูปแบบ : dosread drive [-a] MS-DOS-file > MINIX-file

ตัวเลือก : -a หมายถึง ไฟล์เป็นแบบแอสกี้ไฟล์

หน้าที่ : สำหรับอ่านไฟล์จากดอสลงบนมินิกซ์

ตัวอย่าง : dosread 1 -a input.c > output.c

หมายถึง ให้อ่านไฟล์ input.c เป็นแบบแอสกี้ไฟล์ จากไดร์ฟ 1 ลงใน

ไฟล์ชื่อ output.c

คำสั่ง : doswrite (write a file onto an MS-DOS diskette

รูปแบบ : doswrite [-a] drive DOS-file < MINIX-file

ตัวเลือก : -a หมายถึง ไฟล์เป็นแบบแอสกี้ไฟล์

หน้าที่ : สำหรับเขียนไฟล์ จากมินิกซ์ลงบนดอส

ตัวอย่าง : doswrite -a 1 file1.c < file2.c

หมายถึง ให้เขียนไฟล์ file2.c เป็นแบบแอสกี้ไฟล์ ลงในไดร์ฟ 1 ลงในไฟล์ชื่อ file1.c

คำสั่ง : grep (search a file for lines containing a given pattern)

รูปแบบ : grep [-ensv] pattern [file] ...

ตัวเลือก : -e หมายถึง ให้ค้นหาข้อมูลตามที่กำหนด

-n หมายถึง ให้แสดงหมายเลขบรรทัด

-s หมายถึง แสดงสถานะเท่านั้น ไม่แสดงผลลัพธ์

-v หมายถึง ให้แสดงผลที่ไม่ใช่ข้อมูลที่กำหนด

หน้าที่ : ใช้สำหรับค้นหาคำที่ต้องการในไฟล์ที่ได้กำหนดไว้

ตัวอย่าง : grep -n test *.c

หมายถึง ให้หาคำว่า test ในไฟล์ทุกไฟล์ที่เป็น *.c และให้แสดงหมายเลขบรรทัดด้วย

คำสั่ง : lpr (copy a file to the line printer)

รูปแบบ : lpr [file] ...

ตัวเลือก : ไม่มี

หน้าที่ : ใช้ในการพิมพ์ไฟล์ออกทางเครื่องพิมพ์

ตัวอย่าง : lpr file1 file2

หมายถึง พิมพ์ไฟล์ชื่อ file1 และ file2 ออกทางเครื่องพิมพ์

คำสั่ง : ls (list the contents of a directory

รูปแบบ : ls [-adfgilrst] name ...

ตัวเลือก : -a หมายถึง ให้แสดงไฟล์ทั้งหมดรวมทั้ง . และ ..

-d หมายถึง ไม่แสดงไฟล์ที่เป็นไดเรกทอรี

-f หมายถึง แสดงไฟล์โดยไม่มีการจัดลำดับไฟล์

-g หมายถึง ให้แสดงไฟล์ที่มีกรุ๊ปเหมือนกัน

-i หมายถึง ให้แสดงเบอร์ไอโหนด

-l หมายถึง ให้แสดงผลแบบยาวซึ่งมี โหมด ลิงส์ เจ้าของไฟล์ ขนาด และ เวลา

-r หมายถึง ให้แสดงผลโดยเรียงชื่อไฟล์จากมากไปหาน้อย

-s หมายถึง ให้แสดงขนาดไฟล์เป็นบล็อก

-t หมายถึง ให้แสดงผลโดยเรียงลำดับตามเวลา

หน้าที่ : สำหรับแสดงไฟล์ออกทางจอภาพ

ตัวอย่าง : ls -lis

หมายถึง ให้แสดงไฟล์แบบยาว ไอโหนด และขนาดบล็อก

คำสั่ง : mkdir (make a directory)

รูปแบบ : mkdir directory ...

ตัวเลือก : ไม่มี

หน้าที่ : สำหรับสร้างไดเรกทอรีขึ้นมาใหม่

ตัวอย่าง : mkdir dir

หมายถึง สร้างไดเรกทอรีใหม่ชื่อ dir

คำสั่ง : mkfs (make a file system)

รูปแบบ : mkfs special prototype

ตัวเลือก : -L แสดงลิสติงบนจอภาพ

หน้าที่ : ใช้ทำการสร้างระบบไฟล์ที่เป็นของมินิกซ์ขึ้นเริ่มแรก

ตัวอย่าง : mkfs /dev/fd1 360

หมายถึง ให้สร้างระบบไฟล์ที่มีขนาด 360 บล็อก ลงบนไดร์ฟ 1

คำสั่ง : mount (mount a file system)

รูปแบบ : /etc/mount special file [-r]

ตัวเลือก : -r หมายถึง ระบบไฟล์ที่ถูกเมานท์เข้ามาเป็นแบบอ่านอย่างเดียว

หน้าที่ : ใช้สำหรับเมานท์ระบบไฟล์เข้าสู่ในระบบมินิกซ์ และจะอยู่ตลอดไปจนกว่าจะเมานท์ออกจากระบบ

ตัวอย่าง : /etc/mount /dev/fd1 /user

หมายถึง เมานท์ระบบไฟล์จากไดร์ฟ 1 สู่ระบบมินิกซ์โดยอยู่ในไดเรกทอรี /user

คำสั่ง : mv (move or rename a file)

รูปแบบ : mv file1 file2

mv file ... directory

ตัวเลือก : ไม่มี

หน้าที่ : ใช้สำหรับเปลี่ยนชื่อไฟล์หรือย้ายไฟล์ไปยังไดเรกทอรีอื่น

ตัวอย่าง : mv oldname newname

หมายถึง เปลี่ยนชื่อไฟล์ oldname เป็น newname

mv file1 file2 /user

หมายถึง ย้ายไฟล์ file1 และ file2 ลงในไดเรกทอรี /user

คำสั่ง : passwd (change a login password)

รูปแบบ : passwd [name]

ตัวเลือก : ไม่มี

หน้าที่ : เป็นตัวเปลี่ยนรหัสผ่านของผู้ใช้

ตัวอย่าง : passwd

หมายถึง เปลี่ยนรหัสผ่านของผู้ใช้ในขณะนั้น

passwd ast

หมายถึง เปลี่ยนรหัสผ่านของ ast (ใช้ได้เฉพาะ super-user เท่านั้น)

คำสั่ง : pwd (print working directory)

รูปแบบ : pwd

ตัวเลือก : หมายถึง

หน้าที่ : พิมพ์ชื่อไดเรกทอรีขณะนั้น

ตัวอย่าง : pwd

คำสั่ง : rm (remove a file)

รูปแบบ : rm [-fir] name ...

ตัวเลือก : -f หมายถึง ให้ลบไฟล์โดยปราศจากการถาม

-i หมายถึง ให้มีการถามก่อนไฟล์นั้นจะลบทิ้ง

-r หมายถึง ให้ลบไดเรกทอรีด้วย

หน้าที่ : ลบไฟล์ออกจากระบบ

ตัวอย่าง : rm -i *.c

หมายถึง ให้ลบไฟล์ที่มี .c โดยมีการถามแต่ละไฟล์ว่าจะลบทิ้งหรือไม่

คำสั่ง : rmdir (remove a directory)

รูปแบบ : rmdir directory ...

ตัวเลือก : ไม่มี

หน้าที่ : ลบไดเรกทอรีออกจากระบบ โดยที่ไดเรกทอรีนั้นจะต้องไม่มีไฟล์ใดๆเหลืออยู่

ตัวอย่าง : rmdir /user/foot

หมายถึง ลบไดเรกทอรี /user/foot

คำสั่ง : time (reort how long a command tasks)

รูปแบบ : time command

ตัวเลือก : ไม่มี

หน้าที่ : ใช้สำหรับตรวจสอบเวลาของคำสั่งหรือโปรแกรม ว่ามี real time, user time, and system time เป็นเท่าไร

ตัวอย่าง : time a.out

หมายถึง ตรวจสอบระยะเวลาทำงานของโปรแกรม a.out

คำสั่ง : umount (unmount a mounted file system)

รูปแบบ : /etc/umount spcial

ตัวเลือก : ไม่มี

หน้าที่ : ยกเลิกการเมานท์ของระบบไฟล์ โดยจะต้องไม่อยู่ในไดเรกทอรีที่จะถูกยกเลิกการเมานท์

ตัวอย่าง : /etc/umount /dev/fd1

หมายถึง ทำการยกเลิกการเมานท์ของระบบไฟล์ ในไดร์ฟ 1

คำสั่ง : update (periodically write the buffer cache to disk)

รูปแบบ : /etc/update

ตัวเลือก : ไม่มี

หน้าที่ : ใช้สำหรับการเขียนข้อมูลที่อยู่ในแคชบัฟเฟอร์ลงในดิสค์เพื่อให้ข้อมูลทันสมัยตลอดเวลา

ตัวอย่าง : /etc/update &

หมายถึง เมื่อระบบได้ถูกบูต ตัวโปรแกรม update จะถูกรันแบบแบล็กกราวเพื่อให้ซิงโครนัสกับระบบเพื่อข้อมูลที่ถูกต้อง

ภาคผนวก ช.

การใช้งานอิดิเตอร์มินอิดิต

อิดิเตอร์มินอิดิต

มินอิดิตเป็นโปรแกรมอิดิเตอร์เต็มหน้าจออย่างง่าย (Simple Full-screen Editor) ที่เก็บไฟล์ที่ทำการแก้ไขไว้ในหน่วยความจำ ทำให้มีความเร็วสูง แต่ไม่สามารถแก้ไขไฟล์ที่มีขนาดใหญ่ๆได้ (ไฟล์ขนาดใหญ่ที่สุดที่แก้ไขได้คือ 43K) การแก้ไขไฟล์ที่มีขนาดใหญ่กว่านี้อาจจะทำโดยการแบ่งไฟล์ออกเป็นส่วนๆด้วยคอมมานด์ split หรือเปลี่ยนไปใช้อิดิเตอร์แอลเล่แทน

หน้าจอจะแบ่งเป็นส่วนที่แก้ไขได้ทั้งหมด 24 บรรทัด โดยบรรทัดสุดท้ายจะแสดงสถานะต่างๆ ตำแหน่งปัจจุบันที่จะทำการแก้ไขอยู่ที่ตำแหน่งเคอร์เซอร์ ตัวอักษรปกติที่พิมพ์เข้าไปจะแทรกลงตรงตำแหน่งเคอร์เซอร์ สำหรับปุ่มคอนโทรลต่างๆและปุ่มบนคีย์แพด (keypad) ที่อยู่ด้านขวาของคีย์บอร์ด ใช้ในการเลื่อนตำแหน่งของเคอร์เซอร์และทำฟังก์ชันต่างๆ

คำสั่งที่เกี่ยวกับการจัดการกับคำ เช่น คำสั่งไปที่คำถัดไป, ไปที่คำก่อนหน้า, ลบคำนั้นทิ้ง จะกำหนดความหมายของ คำ (word) คือ กลุ่มตัวอักษรที่เริ่มต้นและปิดท้ายด้วย white space ซึ่งประกอบไปด้วย space, tab, line feed, start of file หรือ end of file เป็นต้น คำสั่งในการลบตัวอักษรและคำสามารถใช้ได้กับรหัส line feed ดังนั้นจึงสามารถต่อข้อความสองบรรทัดเข้าด้วยกันโดยการลบรหัส line feed ของบรรทัดแรกออก

มินอิดิตยังมีการจัดการกับบัฟเฟอร์โดยไม่แสดงบนจอภาพ โดยมีคำสั่งในการลอกข้อความในไฟล์ที่กำลังแก้ไขไปเก็บไว้ในบัฟเฟอร์, จากบัฟเฟอร์ไปเก็บไว้ในไฟล์ หรือเขียนบัฟเฟอร์ไปเก็บไว้ในไฟล์ใหม่ ถ้าเอกสารที่แก้ไขไม่สามารถเขียนกลับไปในดิสค์ เช่น เมื่อดิสค์เต็ม ก็สามารถลอกเอกสารไปเก็บไว้ในบัฟเฟอร์ และเขียนลงไปในไฟล์ในดิสค์อื่นๆ หรืออาจจะออกไปยังเชลล์เพื่อลบไฟล์ที่ไม่จำเป็นออกเพื่อให้ดิสค์มีที่ว่างพอ

บางคำสั่งจะมีการถามให้ตอบ เช่น ชื่อไฟล์, ข้อความที่จะใช้ค้นหา และ คำสั่งที่มีควรจะมีการถามยืนยัน เป็นต้น

คำสั่งหรือปุ่มใดสามารถสั่งให้ทำซ้ำ n รอบได้โดยการพิมพ์ ESC n นำหน้า โดย n เป็นตัวเลขที่แสดงจำนวนรอบ

การค้นหาข้อความ

การค้นหาคำทั้งแบบไปข้างหน้าและค้นย้อนหลัง สามารถค้นเป็นนิพจน์ (expression) โดยมีกฎการค้นเหมือนกับอิดิตเตอร์ ed บนยูนิกซ์ คือ

1. ตัวอักษรปรกติ จะแทนตัวอักษรนั้นๆ
2. เครื่องหมาย . จะแทนตัวอักษรใดๆก็ได้ยกเว้นรหัส line feed
3. เครื่องหมาย ^ จะแทนจุดเริ่มต้นของบรรทัด
4. เครื่องหมาย $ จะแทนจุดสุดท้ายของบรรทัด
5. \c จะแทนตัวอักษร c (ใช้ในการแทนเครื่องหมาย . ^ $ เป็นต้น)
6. [string] จะแทนตัวอักษรใดก็ได้ที่อยู่ใน string
7. [^string] จะแทนตัวอักษรใดก็ได้ที่ไม่อยู่ใน string นั้น
8. [x-y] จะแทนตัวอักษรใดๆ ระหว่าง x และ y เช่น [a-z] เป็นต้น
9. เครื่องหมาย * จะแทนตัวอักษรใดๆกี่ตัวก็ได้ในแพทเทิร์น (pattern)

ตัวอย่างของนิพจน์ต่างๆ มีดังนี้

The boy	แทนข้อความ "The boy"
^$	แทนบรรทัดว่างๆ
^A.*\.$	แทนบรรทัดใดๆที่เริ่มต้นด้วย A, ลงท้ายด้วยเครื่องหมาย .
^[A-Z]*$	แทนบรรทัดใดๆที่มีแต่ตัวอักษรพิมพ์ใหญ่เท่านั้น (หรือไม่มีเลยก็ได้)
[A-ZO-9]	แทนบรรทัดใดๆที่มีตัวอักษรพิมพ์ใหญ่หรือตัวเลข

การแทรกตัวอักษรที่ตรงกับรหัสคอนโทรลต่างๆทำได้โดยการกดปุ่ม ALT ค้างไว้และกดปุ่ม ESC และปล่อยทั้งสองปุ่ม แล้วจิ้งคีย์ปุ่มคอนโทรลนั้น โดยตัวอักษรที่เป็นรหัสจะแสดงเป็นตัวรีเวิร์ส (reverse)

คำสั่งต่างๆในมินิอิดิต

1. การเลื่อนตำแหน่งของเคอร์เซอร์ (Cursor Motion)

ปุ่มลูกศรต่างๆ	เลื่อนเคอร์เซอร์ในทิศทางของลูกศรนั้นๆ
CTRL-A	เลื่อนเคอร์เซอร์ไปที่ตำแหน่งแรกสุดของบรรทัดนั้น
CTRL-Z	เลื่อนเคอร์เซอร์ไปที่ตำแหน่งสุดท้ายของบรรทัดนั้น
CTRL-^	เลื่อนเคอร์เซอร์ไปที่ตำแหน่งบนสุดของจอภาพ
CTRL-_	เลื่อนเคอร์เซอร์ไปที่ตำแหน่งล่างสุดของจอภาพ
CTRL-F	เลื่อนเคอร์เซอร์ไปที่คำถัดไป
CTRL-B	เลื่อนเคอร์เซอร์ไปที่ตัวอักษรแรกของคำก่อน

2. การเลื่อนไปที่ตำแหน่งต่างๆของจอภาพ (Screen Motion)

ปุ่ม Home	ไปที่ตัวอักษรแรกสุดของไฟล์
ปุ่ม End	ไปที่ตัวอักษรสุดท้ายของไฟล์
ปุ่ม PgUp	เลื่อน (scroll) หน้าต่างขึ้น 22 บรรทัด
ปุ่ม PgDn	เลื่อนจอภาพลง 22 บรรทัด
CTRL-U	เลื่อนหน้าต่างขึ้นหนึ่งบรรทัด
CTRL-D	เลื่อนหน้าต่างลงหนึ่งบรรทัด

3. การแก้ไขข้อความ (Modifying Text)

ปุ่ม Del	ลบตัวอักษรที่ตำแหน่งของเคอร์เซอร์
ปุ่ม Backsp	ลงตัวอักษรทางซ้ายของตำแหน่งเคอร์เซอร์หนึ่งตัวอักษร
CTRL-N	ลบคำถัดไป
CTRL-P	ลบคำก่อนหน้า
CTRL-T	ลบตัวอักษรทั้งหมดจากตำแหน่งเคอร์เซอร์ไปยังสุดบรรทัด
CTRL-O	แทรกบรรทัดว่างหนึ่งบรรทัด
CTRL-G	อ่านไฟล์ภายนอกเข้ามาต่อที่ตำแหน่งเคอร์เซอร์

4. บัฟเฟอร์ (Regions)

CTRL-@	เซตตำแหน่งปัจจุบันสำหรับใช้กับปุ่ม CTRL-C และ CTRL-K
CTRL-C	ก็อปปี้ข้อความระหว่างตำแหน่งที่เซตและตำแหน่งเคอร์เซอร์ ลงไปเก็บในบัฟเฟอร์
CTRL-K	เหมือน CTRL-C แต่จะลบข้อความระหว่างตำแหน่งที่เซตและ ตำแหน่งเคอร์เซอร์ทิ้งด้วย
CTRL-Y	ก็อปปี้ข้อความที่เก็บอยู่ในบัฟเฟอร์ลงในตำแหน่งของเคอร์เซอร์
CTRL-Q	เขียนบัฟเฟอร์ไปยังไฟล์ชื่ออื่น

5. อื่นๆ (Miscellaneous)

ปุ่ม numeric +	ค้นหาคำตั้งแต่ตำแหน่งของเคอร์เซอร์ (forward search)
ปุ่ม numeric -	ค้นหาคำย้อนหลังจากตำแหน่งของเคอร์เซอร์ (backward search)
ปุ่ม numeric 5	แสดงสถานะของไฟล์
CTRL-]	ตามด้วย ESC n CTRL-] ไปที่บรรทัดที่ n
CTRL-R	ค้นหาคำและแทนที่ด้วยคำใหม่
CTRL-L	ค้นหาคำและแทนที่ด้วยคำใหม่ภายในบรรทัดปัจจุบันเท่านั้น
CTRL-W	เขียนไฟล์ที่แก้ไขกลับลงไปในดิสค์
CTRL-X	ออกจากอิดิเตอร์
CTRL-S	ฟอร์คโปรเซสเชลล์ (กลับสู่เอดิเตอร์ด้วย CTRL-D)
CTRL-\	ยกเลิกงานที่ทำในขณะนั้นกลับไปรอรับคีย์ใหม่
CTRL-E	วาดหน้าจอใหม่ โดยให้บรรทัดที่มีเคอร์เซอร์อยู่กึ่งกลางจอภาพ
CTRL-V	แก้ไขไฟล์ใหม่

ภาคผนวก ช.

การใช้งานอิดิเตอร์แอลเล่

อิดิเตอร์แอลเล่

แอลเล่ (elle) เป็นโปรแกรมฟูลสกรีนเท็กซ์เอดิเตอร์ (full screen text editor) ที่มีความสามารถสูง โดยเลียนแบบการทำงานคล้ายกับเอดิเตอร์ Emacs บนเครื่องยูนิกซ์ มีคำสั่งหรือฟังก์ชันคีย์ประมาณ 80 คำสั่ง โดยมีคุณสมบัติสำคัญๆ ดังนี้

1. สามารถแก้ไขไฟล์ที่มีขนาดไม่จำกัดได้ โดยขนาดของไฟล์ขึ้นอยู่กับเนื้อที่ของดิสค์
2. เปิดไฟล์สำหรับแก้ไขได้มากที่สุด 2 ไฟล์พร้อมๆกัน
3. ผู้ใช้สามารถกำหนดชนิดของฟังก์ชันคีย์เองได้ (user-defined key binding)
4. สามารถค้นหา (search) คำได้อย่างมีประสิทธิภาพ

ฟังก์ชันคีย์ที่ใช้ในแอลเล่

เนื่องจากจำนวนคำสั่งที่แอลเล่สนับสนุนมีจำนวนมาก เกินกว่าที่จะแม็ปลงในปุ่มคอนโทรลคีย์เพียงปุ่มเดียวได้ ดังนั้นนอกจากคอนโทรลคีย์แล้ว แอลเล่ยังใช้วิธีการกำหนดปุ่ม prefix นำหน้าแล้วจึงตามด้วยฟังก์ชันคีย์ เช่น CTRL-X CTRL-L คือการกดปุ่มคอนโทรลค้างไว้แล้วกดปุ่ม x แล้วตามด้วยปุ่ม L เป็นต้น โดยบางครั้งสามารถกดได้เป็น CTRL-X L หรือ CTRL-x l ก็ถือว่ามีความหมายเหมือนกัน

เนื่องจากคุณสมบัติหนึ่งของแอลเล่ คือ การยอมให้ผู้ใช้กำหนดคีย์สำหรับทำฟังก์ชันต่างๆได้เอง โดยการทำเป็น user profile ซึ่งหมายถึงไฟล์ที่เก็บว่าฟังก์ชันไหนจะกำหนดด้วยปุ่มคีย์ใดบ้าง ซึ่งโปรไฟล์นี้จะต้องผ่านการคอมไพล์ให้อยู่ในรูปของไบนารี ซึ่งแอลเล่จะทำการหาไฟล์โปรไฟล์นี้ที่มีชื่อว่า .ellepro.bi ที่ไดเรกทอรี $HOME ถ้าหาพบก็จะอ่านเข้ามาแทนที่ฟังก์ชันคีย์เดิมของแอลเล่

การใช้แอลเล่เลียนแบบมินอิดิต

เนื่องจากแอลเล่สามารถกำหนดฟังก์ชันคีย์ใหม่ได้ ดังนั้นจึงสามารถกำหนดปุ่มฟังก์ชันคีย์เลียนแบบมินอิดิตได้ โดยการติดตั้งไฟล์ .elIopro.bi ซึ่งเป็นไบนารีฟอร์มของฟังก์ชันคีย์ในมินอิดิต ไว้ในไดเรกทอรีที่ *HOME โดยมีชุดคำสั่งดังนี้

คำสั่งที่เลียนแบบมินอิดิต

1. การเลื่อนตำแหน่งของเคอร์เซอร์ (Cursor Motion)

ปุ่มลูกศรต่างๆ	เลื่อนเคอร์เซอร์ในทิศทางของลูกศรนั้นๆ
CTRL-A	เลื่อนเคอร์เซอร์ไปที่ตำแหน่งแรกสุดของบรรทัดนั้น
CTRL-Z	เลื่อนเคอร์เซอร์ไปที่ตำแหน่งสุดท้ายของบรรทัดนั้น
CTRL-F	เลื่อนเคอร์เซอร์ไปที่คำถัดไป
CTRL-B	เลื่อนเคอร์เซอร์ไปที่ตัวอักษรแรกของคำก่อน

2. การเลื่อนไปที่ตำแหน่งต่างๆของจอภาพ (Screen Motion)

ปุ่ม Home	ไปที่ตัวอักษรแรกสุดของไฟล์
ปุ่ม End	ไปที่ตัวอักษรสุดท้ายของไฟล์
ปุ่ม PgUp	เลื่อน (scroll) หน้าต่างขึ้น 22 บรรทัด
ปุ่ม PgDn	เลื่อนจอภาพลง 22 บรรทัด
CTRL-U	เลื่อนหน้าต่างขึ้นหนึ่งบรรทัด
CTRL-D	เลื่อนหน้าต่างลงหนึ่งบรรทัด
ESC ,	ไปที่ตำแหน่งบนสุดของจอภาพ
CTRL-_	ไปที่ตำแหน่งล่างสุดของจอภาพ

3. การแก้ไขข้อความ (Modifying Text)

ปุ่ม Del	ลบตัวอักษรที่ตำแหน่งของเคอร์เซอร์
ปุ่ม Backsp	ลงตัวอักษรทางซ้ายของตำแหน่งเคอร์เซอร์หนึ่งตัวอักษร
CTRL-N	ลบคำถัดไป
CTRL-P	ลบคำก่อนหน้า
CTRL-T	ลบตัวอักษรทั้งหมดจากตำแหน่งเคอร์เซอร์ไปยังสุดบรรทัด
CTRL-O	แทรกบรรทัดว่างหนึ่งบรรทัด
ESC G	อ่านไฟล์ภายนอกเข้ามาต่อที่ตำแหน่งเคอร์เซอร์ (ตรงกับปุ่ม CTRL-G ในมินิอิดิต)

4. บัฟเฟอร์ (Regions)

CTRL-^	เซตตำแหน่งปัจจุบันสำหรับใช้กับปุ่ม CTRL-C และ CTRL-K
CTRL-C	ก็อปปี้ข้อความระหว่างตำแหน่งที่เซตและตำแหน่งเคอร์เซอร์ลงไปเก็บในบัฟเฟอร์
CTRL-K	เหมือน CTRL-C แต่จะลบข้อความระหว่างตำแหน่งที่เซตและตำแหน่งเคอร์เซอร์ทิ้งด้วย
CTRL-Y	ก็อปปี้ข้อความที่เก็บอยู่ในบัฟเฟอร์ลงในตำแหน่งของเคอร์เซอร์

5. อื่นๆ (Miscellaneous)

ปุ่ม numeric +	ค้นหาคำตั้งแต่ตำแหน่งของเคอร์เซอร์ (forward search)
ปุ่ม numeric -	ค้นหาคำย้อนหลังจากตำแหน่งของเคอร์เซอร์ (backward search)
CTRL-J	ตามด้วย ESC n CTRL-J ไปที่บรรทัดที่ n
CTRL-R	ค้นหาคำและแทนที่ด้วยคำใหม่
CTRL-L	ค้นหาคำและแทนที่ด้วยคำใหม่ภายในบรรทัดปัจจุบันเท่านั้น
CTRL-W	เขียนไฟล์ที่แก้ไขกลับลงไปในดิสค์

CTRL-S	ฟอร์คโปรเซสเซลล์ (กลับสู่เอดิเตอร์ด้วย CTRL-Q)
CTRL-G	ยกเลิกงานที่เอดิเตอร์ทำอยู่ในขณะนั้นและกลับไปรอรับคำสั่งใหม่ (CTRL-\)
CTRL-E	วาดหน้าจอใหม่ โดยให้บรรทัดที่มีเคอร์เซอร์อยู่กึ่งกลางจอภาพ
CTRL-V	แก้ไขไฟล์ใหม่
CTRL-Q	เขียนบัฟเฟอร์เก็บไว้ในไฟล์
ESC X	ออกจากเอดิเตอร์

<u>คำสั่งอื่นๆที่ไม่เหมือนมินอิดิต</u>

1. การเลื่อนตำแหน่งของเคอร์เซอร์ (Cursor Motion)

ESC P	ไปที่พารากราฟถัดไป (พารากราฟที่มีบรรทัดแรกขึ้นต้นด้วยจุด '.')
ESC [	ไปที่พารากราฟก่อนหน้า
ESC .	ย่อหน้า

2. การแก้ไขข้อความ (Modifying Text)

CTRL-\	แทรกตัวอักษรถัดไป (ใช้สำหรับแทรกตัวอักษรคอนโทรลต่างๆ)
ESC =	ลบ white space (ในแนวนอน)
ESC |	ลบบรรทัดว่าง (ในแนวตั้ง)

3. บัฟเฟอร์ (Regions)

ESC M	เซตตำแหน่งพารากราฟปัจจุบัน
ESC ^	เปลี่ยนตำแหน่งเคอร์เซอร์กับตำแหน่งที่เซตพารากราฟ
ESC Y	แทรกข้อความในบัฟเฟอร์ก่อนหน้าบัฟเฟอร์ปัจจุบัน (CTRL-Y จะแทรกข้อความในบัฟเฟอร์ปัจจุบัน)
ESC A	นำบัฟเฟอร์ถัดไปแทรกเข้าไปในบัฟเฟอร์ขณะนั้น

4. การทำมาโครคีย์ (Keyboard Macro)

ESC /	เริ่มบรรทัดปุ่มคีย์
ESC \	จบการบันทึก
ESC *	แสดงมาโครคีย์
ESC E	เรียกใช้มาโครคีย์

5. การจัดการกับหน้าต่าง (Window Management)

^X 1	เข้าสู่โหมดหน้าต่างเดียว
^X 2	เข้าสู่โหมดสองหน้าต่าง
^X L	ขยายขนาดหน้าต่างปัจจุบันให้ใหญ่ขึ้น
^X P	ลดขนาดหน้าต่างปัจจุบันให้เล็กลง
^X N	ไปที่หน้าต่างถัดไป
^X W	ไปที่หน้าต่างใหม่

6. การจัดการกับบัฟเฟอร์ (Buffer Management)

ปุ่ม numeric 5	แสดงรายการไฟล์ปัจจุบันและบัฟเฟอร์ทั้งหมด
ESC B	เลือกบัฟเฟอร์
ESC S	เลือกบัฟเฟอร์ที่มีอยู่จริง
ESC N	เซตให้บัฟเฟอร์มีสภาพเสมือนกับมิได้มีการแก้ไขมาก่อน

7. การจัดการกับตัวอักษรใหญ่/เล็ก (Upper and lower case manipulation)

ESC I	เซตให้ตัวอักษรแรกของคำนั้นเป็นตัวพิมพ์ใหญ่
ESC C	เซตให้คำปัจจุบันเป็นตัวพิมพ์ใหญ่
ESC O	เซตให้คำปัจจุบันเป็นตัวพิมพ์เล็ก
ESC U	เปลี่ยนขอบเขตระหว่างตำแหน่งที่เซตกับเคอร์เซอร์ให้เป็นตัวพิมพ์ใหญ่

ESC L	เปลี่ยนขอบเขตระหว่างตำแหน่งที่เซตกับเคอร์เซอร์ให้เป็นตัวพิมพ์เล็ก

8. อื่นๆ (Miscellaneous)

ESC F	อ่านไฟล์มาเก็บไว้ในบัฟเฟอร์
ESC Z	ค้นหาคำถัดไป
ESC Q	เหมือนกับ CTRL-R แต่จะถามทุกครั้ง
ESC R	รีเซตโปรไฟล์
ESC H	ขอความช่วยเหลือ (help)
ESC ;	สร้างเครื่องหมาย /* */ สำหรับเขียนคอมเมนต์ภาษาซี
^X X	ออกจากอิดิตเตอร์ (เหมือนกับ ESC X)

ข้อแตกต่างที่สำคัญระหว่างมินิอิดิตและเอลเล่ที่เลียนแบบมินิอิดิต

1. การนิยามคำ (word) สำหรับคำถัดไปและคำก่อนหน้า แตกต่างกัน
2. การเซตตำแหน่งใช้ปุ่ม CTRL-^ แทน CTRL-@
3. ใช้ปุ่ม CTRL-G ในการยกเลิกคำสั่ง แทนการใช้ปุ่ม CTRL-\
4. ใช้ปุ่ม CTRL-\ ในการแทรกตัวอักษรคอนโทรลต่างๆ แทนการใช้ปุ่ม ALT
5. ปุ่ม CTRL-E จะปรับหน้าต่างให้เคอร์เซอร์อยู่ตรงกลาง
6. ในการอ่านไฟล์เข้ามาแทรกในอิดิตเตอร์ ให้ใช้ปุ่ม ESC G แทน CTRL-G
7. การไปที่บรรทัดที่ n ให้ใช้ปุ่ม CTRL-J ESC n CTRL-J แทนการใช้ CTRL-[n
8. ออกจากเอลเล่ด้วย CTRL-X CTRL-X และตอบคำถามด้วย 'y'
9. มีคำสั่งเพิ่มเติม ที่จัดการเกี่ยวกับวินโดว์, แก้ไขไฟล์ขนาดใหญ่ขึ้น เป็นต้น

การใช้แอดเล่เลียนแบบ Emacs

1. การเลื่อนตำแหน่งของเคอร์เซอร์ (Cursor Motion)

CTRL-F	ไปข้างหน้าหนึ่งตัวอักษร
CTRL-B	ถอยกลับหนึ่งตัวอักษร
CTRL-H	ถอยกลับหนึ่งตัวอักษร (เหมือนกับ CTRL-B)
ESC F	ไปข้างหน้าหนึ่งคำ
ESC B	ถอยกลับหนึ่งคำ
CTRL-A	ไปที่จุดเริ่มต้นของบรรทัดนั้น
CTRL-E	ไปที่ตำแหน่งสุดท้ายของบรรทัดนั้น
CTRL-N	ไปที่บรรทัดถัดไป
CTRL-P	ไปที่บรรทัดก่อนหน้า
CTRL-V	ไปที่จุดเริ่มต้นของหน้าจอถัดไป
ESC V	ไปที่จุดเริ่มต้นของหน้าจอก่อนหน้า
ESC]	ไปที่พารากราฟถัดไป
ESC [	กลับไปยังพารากราฟก่อน
ESC <	ไปที่จุดเริ่มต้นของบัฟเฟอร์
ESC >	ไปที่จุดสุดท้ายของบัฟเฟอร์

2. การลบตัวอักษรหรือข้อความ (Deletion)

CTRL-D	ลบตัวอักษรที่ตำแหน่งเคอร์เซอร์
DELETE	ลบตัวอักษรก่อนหน้าตำแหน่งเคอร์เซอร์
ESC D	ลบคำข้างหน้า
ESC DEL	ลบคำก่อนหน้า
CTRL-K	ลบจนถึงสุดบรรทัดจากตำแหน่งเคอร์เซอร์
ESC \	ลบช่องว่างระหว่างบรรทัดของเคอร์เซอร์
CTRL-X CTRL-O	ลบบรรทัดว่างรอบๆตำแหน่งเคอร์เซอร์

3. เปลี่ยนตัวพิมพ์เล็ก/ใหญ่ (Case Change)

ESC C	แปลงเป็นคำที่ตัวอักษรแรกเป็นตัวพิมพ์ใหญ่ ที่เหลือเป็นตัวเล็ก
ESC L	แปลงคำถัดไปทั้งคำเป็นตัวพิมพ์เล็ก
ESC U	แปลงคำถัดไปทั้งคำเป็นตัวพิมพ์ใหญ่
CTRL-X CTRL-L	แปลง region ทั้งหมดเป็นตัวพิมพ์เล็ก
CTRL-X CTRL-U	แปลง region ทั้งหมดเป็นตัวพิมพ์ใหญ่

4. การค้นหา (Searching)

ถ้าไม่บอกข้อความ จะใช้ข้อความครั้งที่แล้ว

CTRL-S	ค้นไปข้างหน้า และแสดงข้อความ "I-search:"
CTRL-R	ค้นย้อนหลัง และแสดงข้อความ "R-search:"

ระหว่างการค้นหา ตัวอักษรต่อไปนี้ มีผลดังนี้

"normal" chars	- เริ่มค้นหาแบบปรกติ
^G	- ยกเลิกการค้นไปข้างหน้าและกลับไปยังจุดเริ่มต้น
DEL	- ลบตัวอักษรครั้งที่แล้วและกลับไปยังจุดค้นพบครั้งก่อน
^S, ^R	- ค้นหาซ้ำอีกครั้ง (หรือเปลี่ยนทิศการค้นหา)
ESC หรือ CR	- ออกจากการค้นไปข้างหน้ากลับไปที่ตำแหน่งขณะนั้น

ESC %	ค้นและแทนที่แบบถามตอบ ตอบด้วย "?" เพื่อดูตัวเลือก
CTRL-X %	ค้นและแทนที่โดยไม่มีการถามตอบเมื่อพบคำนั้น

5. กำหนด region (Marking Areas)

CTRL-^	เซตตำแหน่ง
CTRL-X CTRL-X	เปลี่ยนตำแหน่งเคอร์เซอร์และตำแหน่งที่เซตไว้
ESC H	เซตพารากราฟตั้งแต่จุดเริ่มต้นของพารากราฟ
CTRL-W	ลบ region
ESC W	ก็อปปี้ region เหมือนกับการใช้ CTRL-W CTRL-Y

CTRL-Y	แทรกข้อความที่ถูกลบครั้งล่าสุดไปที่ตำแหน่งเคอร์เซอร์
ESC Y	แทรกข้อความก่อนครั้งล่าสุดไปที่ตำแหน่งเคอร์เซอร์
ESC CTRL-W	นำข้อความไปต่อกับข้อความที่ถูกลบครั้งล่าสุด

6. เติมข้อความ (Filling Text)

ESC Q	เติมพารากราฟด้วยความของคอลัมน์ที่เติม
ESC G	เติม region
CTRL-X F	เซตคอลัมน์ที่เติม เพื่อนำไปใช้ใน ESC Q
CTRL-X .	เซตข้อความที่เติม
CTRL-X T	สลับโหมดในการเติมอัตโนมัติ

7. หน้าต่าง (Windows)

CTRL-X 2	แบ่งหน้าจอเป็น 2 หน้าต่าง
CTRL-X 1	กำหนดหน้าจอเป็นหน้าต่างเดียว
CTRL-X O	ไปที่หน้าต่างถัดไป
CTRL-X ^	ขยายขนาดหน้าต่างปัจจุบัน

8. บัฟเฟอร์ (Buffers)

CTRL-X CTRL-F	อ่านไฟล์มาเก็บไว้ในบัฟเฟอร์
CTRL-X B	เลือกบัฟเฟอร์ที่จะแก้ไขหรือจองบัฟเฟอร์สำหรับไฟล์ใหม่
CTRL-X CTRL-B	แสดงชื่อไฟล์ที่เก็บไว้ในบัฟเฟอร์ทั้งหมด
CTRL-X K	ลบบัฟเฟอร์
ESC ~	เซตให้บัฟเฟอร์ยังไม่มีการแก้ไข (แม้จะมีการแก้ไขจริง)
CTRL-X CTRL-M	สลับโหมด EOL (แฟลกของแต่ละบัฟเฟอร์)

9. การทำมาโครคีย์ (Keyboard Macro)

CTRL-X (	เริ่มบรรทัดปุ่มคีย์
CTRL-X)	จบการบันทึก
CTRL-X *	แสดงมาโครคีย์
CTRL-X E	เรียกใช้มาโครคีย์

10. ไฟล์ (Files)

CTRL-X CTRL-I	อ่านไฟล์มาแทรกไว้ที่ตำแหน่งเคอร์เซอร์
CTRL-X CTRL-R	อ่านไฟล์ใหม่เข้ามาสู่บัฟเฟอร์ปัจจุบัน
CTRL-X CTRL-V	เหมือนกับ ^X ^R
CTRL-X CTRL-W	เขียนบัฟเฟอร์ไปยังไฟล์
CTRL-X CTRL-S	เก็บไฟล์ลงในดิสค์
CTRL-X CTRL-E	เขียน region ไปยังไฟล์ชื่อใหม่

11. อื่นๆ (Miscellaneous)

CTRL-X CTRL-Z	ออกจากแอลเล่
CTRL-X !	เรียกเชลล์ (กลับโดยใช้ CTRL-D)
CTRL-O	แทรกบรรทัดว่างหนึ่งบรรทัด
LINEFEED	เหมือนกับการพิมพ์ RETURN ตามด้วย TAB
CTRL-U	ทำให้คำสั่งถัดไปทำซ้ำสี่รอบ
CTRL-U number	ทำให้คำสั่งถัดไปทำซ้ำ number รอบ
ESC number	เหมือนกับ CTRL-U number
CTRL-L	วาดหน้าจอใหม่
CTRL-U CTRL-L	วาดหน้าจอใหม่เมื่อมีบรรทัดที่มีเคอร์เซอร์อยู่
^U n ^L	เปลี่ยนหน้าต่างเพื่อให้เคอร์เซอร์อยู่ที่บรรทัดที่ n
CTRL-Q	แทรกตัวอักษรถัดไปลงในบัฟเฟอร์ ไม่ว่าจะตรงกับคำสั่งใด

CTRL-G	ยกเลิกการถามให้ตอบ
ESC ;	แทรกคอมเมนต์ภาษาซี /* */
ESC I	ย่อหน้าให้ตรงกับบรรทัดที่แล้ว
ESC M	ไปที่ตำแหน่งสุดท้ายของย่อหน้าในบรรทัดนั้น
CTRL-_	อธิบายคำสั่งต่างๆถ้ามีฐานข้อมูลเก็บคำอธิบายอยู่

12. คำสั่งควบคุมที่ไม่ควรใช้

CTRL-\	คำสั่งเฉพาะสำหรับดีบักโปรแกรม
CTRL-^	คำสั่งเฉพาะสำหรับดีบักโปรแกรม
CTRL-C	ไม่ใช้
CTRL-Z	ไม่ใช้
CTRL-]	ไม่ใช้

การติดตั้งแอลเลในมินิกซ์

ใช้ไฟล์ต่างๆดังนี้

1) elle	(โปรแกรมแอลเล่อิดิตเตอร์)
2) ellec	(โปรไฟล์คอมไพเลอร์)
3) .ellepro.e	(ไฟล์ที่เก็บโปรไฟล์)
4) .ellepro.bi	(ไฟล์ที่เก็บโปรไฟล์ที่คอมไพล์แล้ว)
5) help.dat	(ฐานข้อมูลแสดงความช่วยเหลือ)

โดยมีขั้นตอนในการติดตั้งดังนี้

1. ตรวจให้แน่ใจว่า ในไฟล์ /etc/termcap มีอยู่จริงและมีรายการของ minix อยู่
2. เซตสภาพแวดล้อม (Environment) เพื่อให้มีตัวแปร TERM=minix อยู่
3. ก็อปปี้โปรแกรม elle และ ellec (ไม่จำเป็น) ลงในไดเรกทอรี /bin หรือ /usr/bin

4. ก็อปปี้โปรแกรม hekp.dat (ไม่จำเป็น) ลงในไฟล์ /usr/src/elle/help.dat
5. ถ้าต้องการให้แอลเล่ทำงานเหมือนมินิอิดิต ก็อปปี้ไฟล์ .ellepro.bi ลงในไดเรกทอรี $HOME
6. เข้าสู่แอลเล่ โดยพิมพ์ 'elle ชื่อไฟล์'

ขั้นตอนในการสร้างโปรไฟล์ของตัวเอง

1. แก้ไขไฟล์ .ellepro.e ตามความต้องการ
2. ก็อปปี้ไฟล์ .ellepro.e ไว้ในไดเรกทอรี $HOME
3. คอมไพล์โปรไฟล์โดยเรียก ellec -Profile
4. ตรวจดูว่ามีไฟล์ $HOME/.ellepro.bi เกิดขึ้นหรือไม่

ภาคผนวก ฉ,

สรุปซิสเต็มคอลล์ระบบภาษาไทย

Vclrkey

ชื่อฟังก์ชัน	Vclrkey - Clear keyboard input queue
การใช้งาน	void Vclrkey(void)
คำอธิบาย	ทำการเคลียร์อินพุทบัฟเฟอร์ทั้งหมดที่ยังค้างอยู่ในอินพุทคิว
ค่าที่ส่งกลับ	-

Vclrscr

ชื่อฟังก์ชัน	Vclrscr - Clear screen
การใช้งาน	void Vclrscr(void)
คำอธิบาย	ทำการลบหน้าจอ
ค่าที่ส่งกลับ	-

Vdmodectrl

ชื่อฟังก์ชัน Vdmodectrl - Display mode control

การใช้งาน void Vdmodectrl(int flag)

คำอธิบาย ใช้กำหนดว่า จะสามารถใช้คีย์ปรกติของมินิกซ์ในการเปลี่ยนโหมดแสดงผลได้หรือไม่ โดยค่าของ flag ที่ใช้ได้คือ

_ENABLE - อนุญาตให้ใช้คีย์ปรกติได้ คือ คีย์ F8

_DISABLE - ไม่สามารถเปลี่ยนโหมดแสดงผลโดยใช้คีย์ปรกติได้

การใช้ประโยชน์ของฟังก์ชันนี้ คือ โปรแกรมสามารถกำหนดคีย์ในการเปลี่ยนโหมดเองได้ โดยการใช้คำสั่ง Vdmodectrl(_DISABLE) ก่อน และเมื่อมีการกดคีย์ที่โปรแกรมกำหนด ก็ใช้คำสั่ง Vsetdmode(mode) เพื่อทำการเซ็ตโหมด สำหรับโปรแกรมที่ใช้งานได้โหมดใดโหมดหนึ่งเท่านั้น ก็สามารถบังคับโหมดได้เช่นกัน

ค่าที่ส่งกลับ -

Vgetcolor

ชื่อฟังก์ชัน Vgetcolor - Get attribute or color

การใช้งาน int Vgetcolor(void)

คำอธิบาย ใช้ในการอ่านค่าสี หรือ แอตทริบิวต์

ค่าที่ส่งกลับ สี หรือแอตทริบิวต์ ปัจจุบัน

Vgetdmode

ชื่อฟังก์ชัน Vgetdmode - Get display mode

การใช้งาน int Vgetdmode(void)

คำอธิบาย ใช้ในการตรวจสอบว่า ขณะนี้ แสดงผลอยู่ในโหมดใด

ค่าที่ส่งกลับ _17LINES - สำหรับการแสดงผล 17 บรรทัด จัดระดับ

_25LINES - สำหรับการแสดงผล 25 บรรทัด ไม่จัดระดับ

Vgetkmode

ชื่อฟังก์ชัน Vgetkmode - Get keyboard mode

การใช้งาน int Vgetkmode(void)

คำอธิบาย ใช้ในการตรวจสอบว่า ขณะนี้ โหมดของคีย์บอร์ด เป็นภาษาใด

ค่าที่ส่งกลับ IN_THAI - ถ้าคีย์บอร์ด อยู่ในโหมดรับภาษาไทย

IN_ENG - ถ้าคีย์บอร์ด อยู่ในโหมดรับภาษาอังกฤษ

Vgetx

ชื่อฟังก์ชัน Vgetx - Get cursor 's column

การใช้งาน int Vgetx(void)

คำอธิบาย ใช้อ่านค่าตำแหน่งคอลัมภ์ปัจจุบันของเคอร์เซอร์

ค่าที่ส่งกลับ ค่าคอลัมภ์ปัจจุบันของเคอร์เซอร์

Vgety

ชื่อฟังก์ชัน	Vgety - Get cursor 's row
การใช้งาน	int Vgety(void)
คำอธิบาย	ใช้อ่านค่าตำแหน่งแถวปัจจุบันของเคอร์เซอร์
ค่าที่ส่งกลับ	ค่าแถวปัจจุบันของเคอร์เซอร์

Vgotoxy

ชื่อฟังก์ชัน	Vgotoxy - Move cursor to (x,y)
การใช้งาน	void Vgotoxy(int x,int y)
คำอธิบาย	เลื่อนตำแหน่งเคอร์เซอร์ไปที่ตำแหน่ง (x,y) โดยตำแหน่งมุมบนซ้ายสุดคือ ตำแหน่ง (0,0)
ค่าที่ส่งกลับ	-

Vkmodectrl

ชื่อฟังก์ชัน Vkmodectrl - Keyboard mode control

การใช้งาน void Vkmodectrl(int flag)

คำอธิบาย ใช้กำหนดว่า จะสามารถใช้คีย์ปรกติของมินิกซ์ในการเปลี่ยนโหมดอินพุท ได้หรือไม่ โดยค่าของ flag ที่ใช้ได้คือ

ENABLE - อนุญาตให้ใช้คีย์ปรกติได้ คือ คีย์ F10

DISABLE - ไม่สามารถเปลี่ยนโหมดอินพุท โดยใช้คีย์ปรกติได้

การใช้ประโยชน์ของฟังก์ชันนี้ คือ โปรแกรมสามารถกำหนดคีย์ในการเปลี่ยนโหมดเองได้ โดยการใช้คำสั่ง Vkmodectrl(_DISABLE) ก่อน และเมื่อมีการกดคีย์ที่โปรแกรมกำหนด ก็ใช้คำสั่ง Vsetkmode(mode) เพื่อทำการเซ็ตโหมด สำหรับโปรแกรมที่ใช้งานได้โหมดใดโหมดหนึ่งเท่านั้น ก็สามารถบังคับโหมดได้เช่นกัน

ค่าที่ส่งกลับ -

Vkeypressed

ชื่อฟังก์ชัน Vkeypressed - Check for keypressed

การใช้งาน int Vkeypressed(void)

คำอธิบาย ตรวจสอบว่ามีการกดคีย์เกิดขึ้นหรือไม่ โดยจะยังไม่อ่านค่าคีย์ที่กดเข้ามา

ค่าที่ส่งกลับ TRUE - ถ้ามีการกดคีย์เกิดขึ้น

FALSE - ถ้ายังไม่มีการกดคีย์เลย

Vline

ชื่อฟังก์ชัน Vline - Draw line

การใช้งาน void Vline(int x1,int y1,int x2,int y2,int color)

คำอธิบาย ทำการลากเส้นตรงสี color จากจุด (x1,y1) ไปยังจุด (x2,y2) โดยตำแหน่งของจุดทั้งหมด จะกำหนดเป็นพิกเซล โดยพิกเซลมุมบนซ้ายสุด จะเป็นตำแหน่ง (0,0)

ค่าที่ส่งกลับ -

Vresetfont

ชื่อฟังก์ชัน Vresetfont - Reset installed font

การใช้งาน void Vresetfont(int fontnum);

คำอธิบาย ยกเลิกการใช้ฟอนต์หมายเลข fontnum ที่ได้เซตไว้โดยฟังก์ชัน Vsetfont

ค่าที่ส่งกลับ -

Vscrdown

ชื่อฟังก์ชัน Vscrdown - Scroll screen down

การใช้งาน void Vscrdown(int x1,int y1,int x2,int y2)

คำอธิบาย ใช้ทำการเลื่อนหน้าจอภายในหน้าต่างสี่เหลี่ยมที่กำหนดดังรูป ลงหนึ่งบรรทัด

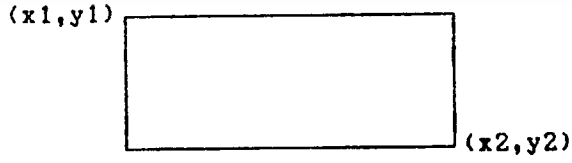
ค่าที่ส่งกลับ -

Vscrup

ชื่อฟังก์ชัน Vscrup - Scroll screen up

การใช้งาน void Vscrup(int x1,int y1,int x2,int y2)

คำอธิบาย ใช้ทำการเลื่อนหน้าจอภายในหน้าต่างสี่เหลี่ยมที่กำหนดดังรูป ขึ้นหนึ่งบรรทัด



ค่าที่ส่งกลับ -

Vsetcolor

ชื่อฟังก์ชัน Vsetcolor - Set attribute or color

การใช้งาน void Vsetcolor(int attr)

คำอธิบาย เซ็ตลิของการแสดงผล ให้เป็นไปตามตัวแปร attr

ค่าที่ส่งกลับ -

Vsetfont

ชื่อฟังก์ชัน Vsetfont - Install user font

การใช้งาน int Vsetfont(int res, int fontnum, int offset)

คำอธิบาย ติดตั้งฟอนท์ของผู้ใช้เอง เพื่อสามารถเรียกใช้งานได้เพิ่มขึ้น จากฟอนท์เดิมที่มินิกซ์มีให้อยู่แล้ว โดยกำหนดโหมดของจอภาพ res ว่าเป็น 17หรือ 25 บรรทัด และหมายเลขของฟอนท์ที่จะติดตั้งระหว่าง 0 ถึง 9 (ฟอนท์หมายเลข 0 และ 1 มีใช้ในระบบอยู่แล้ว)

ค่าที่ส่งกลับ ถ้าติดตั้งได้สำเร็จจะส่งค่าศูนย์กลับ แต่ถ้าไม่สำเร็จจะส่งค่าไม่เท่ากับศูนย์

Vsetdmode

ชื่อฟังก์ชัน Vsetdmode - Set display mode

การใช้งาน void Vsetdmode(int mode)

คำอธิบาย ใช้ในการเซ็ตโหมดของการแสดงผล ให้เป็นตามตัวแปร mode โดยค่าที่ใช้ได้คือ

_17LINES - เซ็ตเป็นการแสดงผล 17 บรรทัด จัดระดับ

_25LINES - เซ็ตเป็นการแสดงผล 25 บรรทัด ไม่จัดระดับ

ค่าที่ส่งกลับ -

Vsetkmode

ชื่อฟังก์ชัน Vsetkmode - Set keyboard mode

การใช้งาน void Vsetkmode(int mode)

คำอธิบาย ใช้เซ็ตคีย์บอร์ด ให้เป็นการรับอินพุทเป็นตามตัวแปร mode ซึ่งค่าที่ใช้ได้ คือ

IN_THAI - สำหรับอินพุทภาษาไทย

IN_ENG - สำหรับอินพุทภาษาอังกฤษ

ค่าที่ส่งกลับ -

Vusefont

ชื่อฟังก์ชัน Vusefont - Use installed font

การใช้งาน void Vusefont(int fontnum)

คำอธิบาย ใช้ฟอนต์ที่ได้ติดตั้งโดย Vsetfont โดยตัวอักษรที่จะแสดงต่อไป จะใช้ฟอนต์ที่เก็บอยู่ในฟอนต์นี้

Vxyac

ชื่อฟังก์ชัน Vxyac - Put character at (x,y) with attribute

การใช้งาน void Vxyac(int x,int y,int attr,char ch)

คำอธิบาย เขียนตัวอักษร ch พร้อมแอตทริบิวต์ attr ออกหน้าจอ ที่ตำแหน่ง (x,y)

ค่าที่ส่งกลับ -

Vxyas

ชื่อฟังก์ชัน Vxyas - Put string at (x,y) with attribute

การใช้งาน void Vxyas(int x,int y,int attr,char *str)

คำอธิบาย เขียนข้อความ str พร้อมแอตทริบิวต์ attr ออกหน้าจอ ที่ตำแหน่ง (x,y)

ค่าที่ส่งกลับ จำนวนตัวอักษรที่ได้แสดงผลไป

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ สามารถสำเร็จได้ ด้วยความร่วมมือ และ ความช่วยเหลือจากหลาย ๆ ฝ่าย ทางคณะผู้จัดทำต้องขอขอบคุณ อาจารย์ที่ปรึกษา อ.บุญธีร์ เครือตราชู ที่คอยให้คำปรึกษา และคอยสอบถามความก้าวหน้าด้วยความเป็นห่วง ขอขอบคุณ อาจารย์ทุก ๆ ท่านที่ได้ให้ความช่วยเหลือ ขอบคุณเพื่อน ๆ ที่คอยเป็นกำลังใจ และท้ายที่สุดนี้ ขอขอบคุณเพื่อนร่วมงาน ที่ได้ร่วมมือกันทำงานจนงานสำเร็จไปด้วยดี

บรรณานุกรม

1. Andrew S. Tanenbaum, "OPERATING SYSTEMS: Design and Implementation", Prentice-Hall International Inc., 719 p.,1987.

2. George J. Sutty, "Programmer's guide to the EGA/VGA", Brady Books, 512 p.,1988.

3. Borland International, "Turbo C Reference guide", 216 p.,1987.

4. ทวีศักดิ์ กออนันตกุล, "ข้อกำหนดร่วมเพื่อการเขียนโปรแกรมซึ่งแสดงผลเป็นภาษาไทย", 116 หน้า,2533.