การพัฒนาเวอร์ชวลแมชีนขนาดเล็กบนอุปกรณ์บอร์ดคอม86

KVM on Com86 Board

โดย

นาย ขจร เจียรนัยพานิชย์

นางสาว จริยาพร บุญสังข์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

การพัฒนาเวอร์ชวลแมชีนขนาดเล็กบนอุปกรณ์บอร์ดคอม86

KVM on Com86 Board

โดย

นาย ขจร เจียรนัยพานิชย์

นางสาว จริยาพร บุญสังข์

อาจารย์ที่ปรึกษา

ผศ. อภิเนตร อูนากูล

อ. ดุสิต นิยะโต

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ ปีการศึกษา 2545

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาเวอร์ชวลแมชีนขนาดเล็กบนอุปกรณ์บอร์ดคอม86

KVM on Com86 Board

ผู้จัดทำ

1. นาย ขจร เจียรนัยพานิชย์ รหัสประจำตัว 42010033
2. นางสาว จริยาพร บุญสังข์ รหัสประจำตัว 42010046

อาจารย์ที่ปรึกษา

(ผศ. อภิเนตร อูนากูล)

อาจารย์ที่ปรึกษา

(อ. ดุสิต นิยะโต)

การพัฒนาเวอร์ชวลแมชีนขนาดเล็กบนอุปกรณ์บอร์ดคอม86

นายขจร เจียรนัยพานิชย์ 42010033
นางสาวจริยาพร บุญสังข์ 42010046
ผศ.อภิเนตร อูนากูล อาจารย์ที่ปรึกษา
อ.ดุสิต นิยะโต อาจารย์ที่ปรึกษา

ปีการศึกษา 2545

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ มีเนื้อหาเกี่ยวกับจาวาไมโครอิดิชั่น (Java Micro Edition) หรือที่เรียกกันสั้นๆ ว่า เจทูเอ็มอี (J2ME) ของบริษัท ซัน ไมโครซิสเต็มส์ จำกัด และมีเนื้อหาเกี่ยวกับอุปกรณ์คอม86 ที่เป็นระบบฝังตัว (Embeded System) โดยมีรายละเอียดที่สำคัญเกี่ยวกับโครงสร้างและหลักการทำงานของเวอร์ชวลแมชีนขนาดเล็ก (KVM) ซึ่งเป็นจาวาเวอร์ชวลแมชีนที่มีขนาดเล็ก ตลอดจนวิธีการพัฒนาและติดตั้งเวอร์ชวลแมชีนขนาดเล็กลงบนอุปกรณ์คอม86 โดยได้ทำการดัดแปลงให้เวอร์ชวลแมชีนขนาดเล็กสามารถทำงานบนระบบปฏิบัติการดอส (DOS) ที่อยู่บนอุปกรณ์คอม86 ได้ ดังนั้น เมื่อพัฒนาเสร็จแล้ว อุปกรณ์คอม86 จะสามารถใช้งานจาวาแอพพลิเคชั่น (Java Application) ได้ โดยที่ความสามารถของโปรแกรมจาวานั้นจะขึ้นอยู่กับข้อจำกัดของอุปกรณ์คอม86 ด้วย การพัฒนานี้จึงทำให้สามารถนำอุปกรณ์คอม86 ไปประยุกต์ใช้งานได้หลากหลายและมีความสามารถมากยิ่งขึ้น

KVM on Com86 Board

MR. KHAJORN CHIARANAIPANICH
MRS. JARIYAPORN BOONSANG
Asst.Prof.APINETR UNAGUL Advisor
MR.DUSIT NIYATO

Abstract

This thesis is to study Java Micro Edition (J2ME) of Sun Microsystems and COM86 device which is an embedded system. We focused on the structure and methodology of a small JAVA virtual machine named K-Virtual Machine (KVM). Moreover, we studied how to implement and install K-Virtual Machine on COM86 device. We modified K-Virtual Machine to be able to run on COM86 DOS operation system. Consequently, COM86 device is capable to run JAVA application. However, the ability of JAVA application is still up to COM86 device constrains. In conclusion, this development helps bring COM86 device to be implemented in various applications.

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลงได้ด้วยดี ต้องขอขอบคุณผู้ให้การช่วยเหลือทุกท่าน โดยเฉพาะอาจารย์ที่ปรึกษา อาจารย์ คุสิต นิยะโต และผู้ช่วยศาสตราจารย์ อภิเนตร อูนากูล ที่ให้คำแนะนำและชี้แนะแนวทางในการศึกษาค้นคว้าและทำการพัฒนาโครงงานนี้ จนสำเร็จลุล่วงไปได้ด้วยดี

ขอขอบคุณเพื่อนๆ พี่ๆ และน้องๆ ทุกคนที่ได้ให้กำลังใจ ให้คำปรึกษา แนะนำและเอื้อเฟื้อข้อมูลในเรื่องต่างๆ ตลอดมา

สุดท้ายนี้ ขอขอบคุณคุณพ่อ และคุณแม่ ที่เลี้ยงดูเรามา ให้กำลังใจ รวมถึงทุนทรัพย์ในการทำปริญญานิพนธ์นี้จนเสร็จสิ้นสมบูรณ์ลงได้

ปริญญานิพนธ์นี้ ขอมอบแด่ผู้มีพระคุณทุกท่าน

ผู้จัดทำ

สารบัญ

หน้า

สารบัญตาราง

สารบัญภาพ

ภาพที่ หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันนี้ ระบบเทคโนโลยีสารสนเทศและระบบคอมพิวเตอร์มีการพัฒนาไปอย่างรวดเร็วมาก ภาษาที่ใช้ในการเขียนโปรแกรมก็ได้มีผู้คิดค้นและพัฒนากันอย่างหลากหลาย จาวาก็เป็นภาษาหนึ่งที่กำลังได้รับความสนใจเป็นอย่างมากในปัจจุบัน เพราะข้อดีของจาวาที่สามารถสร้างโปรแกรมในเครื่องหนึ่งแล้วสามารถนำไปทำงานบนเครื่องคอมพิวเตอร์เครื่องอื่นๆ ที่มีระบบแตกต่างกันได้ และจาวาก็เป็นภาษาที่เป็นการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) ซึ่งทำให้สามารถเขียนแอพพลิเคชั่นที่มีความซับซ้อนมากๆ ได้ จึงทำให้จาวาได้รับความนิยมอย่างแพร่หลายมากขึ้น อย่างไรก็ตาม อุปกรณ์ใดๆที่จะใช้งานภาษาจาวาได้ จำเป็นที่จะต้องมีจาวาเวอร์ชวลแมชีนติดตั้งอยู่บนอุปกรณ์นั้นก่อนเสมอ

นอกจากจาวาแล้ว ในปัจจุบันเทคโนโลยีที่เป็น สิ่งที่อยู่ในความสนใจของคนทั่วไปก็คือ เทคโนโลยีแบบไร้สาย (Wireless Technology) ในขณะนี้ได้มีการพัฒนาไปอย่างมากและมีแนวโน้มว่าต่อไป อุปกรณ์ต่างๆ ส่วนใหญ่ก็จะพัฒนาไปเป็นแบบนี้ เพราะสามารถนำไปประยุกต์ใช้งานได้หลากหลายยิ่งขึ้น อุปกรณ์บอร์ดคอม86 ก็เป็นอุปกรณ์หนึ่งที่พัฒนาขึ้นมาให้เป็นแบบไร้สาย เป็นระบบฝังตัว (Embeded System) ซึ่งเป็นเหมือนกับคอมพิวเตอร์ล่องหน สามารถนำไปใช้งานได้แพร่หลาย เหมือนอย่างคอมพิวเตอร์ทั่วไป แต่ในตอนนี้ อุปกรณ์บอร์ดคอม86 ยังไม่สามารถใช้งานจาวาแอพพลิเคชั่นได้ ดังนั้น การนำเอาภาษาจาวามาพัฒนาและติดตั้งลงบนอุปกรณ์บอร์ดคอม86 ก็จะส่งผลให้เกิดประโยชน์มากยิ่งขึ้นอีก ด้วยเหตุนี้ ก่อนที่จะนำเอาภาษาจาวามาใช้งานได้ ต้องนำเอาเวอร์ชวลแมชีนมาติดตั้งก่อน ซึ่งเราเลือกใช้เวอร์ชวลแมชชีนขนาดเล็กที่มีชื่อว่าเควีเอ็ม(KVM) นั่นเอง ด้วยเหตุนี้ จึงทำให้เกิดโครงการพัฒนาเวอร์ชวลแมชีนขนาดเล็กบนอุปกรณ์บอร์ดคอม86 ขึ้นมา

1.2 วัตถุประสงค์

- เพื่อศึกษาโครงสร้างของจาวาไมโครเอดิชั่น (J2ME)
- เพื่อศึกษาการทำงานของโครงสร้างภายในของเวอร์ชวลแมชีนขนาดเล็ก (KVM)
- เพื่อพัฒนาให้สามารถติดตั้งเวอร์ชวลแมชีนขนาดเล็ก (KVM) ลงบนอุปกรณ์บอร์ดคอม86 (COM86) ที่ใช้ระบบปฏิบัติการดอส(DOS)ได้

1.3 ประโยชน์ที่จะได้รับ

- ทำให้ทราบวิธีการติดตั้งเวอร์ชวลแมชีนขนาดเล็ก (KVM) ลงบนระบบปฏิบัติการดอส และสามารถพัฒนาต่อให้ระบบปฏิบัติการดอสใช้งานจาวาไมโครเอดิชั่นได้
- สามารถนำความรู้และวิธีการติดตั้งเวอร์ชวลแมชีนขนาดเล็ก(KVM) ไปประยุกต์ใช้ในการนำไปติดตั้งบนอุปกรณ์อื่นๆ ได้อีกต่อไป

- ทำให้สามารถนำอุปกรณ์บอร์ดคอม86 ไปใช้งานต่างๆ ได้หลากหลายมากยิ่งขึ้น ซึ่งอุปกรณ์บอร์ดคอม86 เป็นระบบฝังตัว(Embedded System) คือ ระบบที่มีเหมือนมีคอมพิวเตอร์ขนาดเล็กอยู่ภายใน จึงสามารถนำอุปกรณ์บอร์ดคอม86 ไปใช้ประโยชน์ได้อย่างมากมาย ดังนี้
  - ใช้ในเครื่องใช้ไฟฟ้าต่างๆ ในบ้าน เช่น ตู้เย็น เตาไมโครเวฟ โทรศัพท์ เครื่องซักผ้า เป็นต้น ทำให้อุปกรณ์ต่างๆ มีความฉลาดมากขึ้น
  - ใช้ในอุปกรณ์ต่างๆ ในสำนักงานเช่น โปรเจคเตอร์ เครื่องถ่ายเอกสาร เป็นต้น
  - ใช้ในระบบควบคุมโรงงานอัตโนมัติ เช่น ระบบการควบคุมและตรวจสอบการทำงานจากระยะไกล ระบบตรวจวัดข้อมูลต่างๆ เป็นต้น
  - ใช้ในระบบควบคุมอาคารอัตโนมัติ โดยใช้ในการควบคุมอุปกรณ์และระบบต่างๆ ภายในอาคาร เช่น ระบบไฟส่องสว่าง ระบบปรับอากาศ ระบบรักษาความปลอดภัย เป็นต้น

จากการนำระบบฝังตัวไปใช้กันอย่างแพร่หลาย จะเห็นได้ว่า ถ้าอุปกรณ์บอร์ดคอม86 มีความสามารถในการทำงานภาษาจาวาแล้ว ยิ่งจะทำให้ความสามารถในการทำงานมีมากยิ่งขึ้น และสามารถที่จะพัฒนาให้เกิดโปรแกรมประยุกต์ที่มีความสามารถหลากหลายยิ่งขึ้น เพราะจะได้ใช้ข้อดีของภาษาจาวาที่เป็นการทำงานแบบโปรแกรมเชิงวัตถุ (Object-Oriented Programming) ที่ทำให้สามารถทำงานในแบบที่มีความซับซ้อนมากๆ ได้ ดังนั้น จึงมีประโยชน์ในการทำให้เกิดการพัฒนาทางด้านระบบฝังตัวมากยิ่งขึ้น

1.4 เป้าหมายของโครงงาน

- พัฒนาเวอร์ชวลแมชีนขนาดเล็ก (KVM) ให้ทำงานบนระบบปฏิบัติการดอสของอุปกรณ์บอร์ดคอม86 ได้

1.5 ขอบเขตของโครงงาน

- แก้ไขโค้ดของเวอร์ชวลแมชีนขนาดเล็ก (KVM) เพื่อให้สามารถติดตั้งและทำงานบนระบบปฏิบัติการดอสได้
- ศึกษาและทำความเข้าใจโครงสร้างและหน้าที่การทำงานในส่วนต่างๆ ของเวอร์ชวลแมชีนขนาดเล็ก (KVM)

1.5 ขั้นตอนการดำเนินงาน

- ศึกษาโครงสร้างทั้งหมดของจาวาไมโครเอดิชั่น
- ศึกษาโครงสร้าง วิธีการทำงาน และข้อจำกัดต่างๆ ของอุปกรณ์บอร์ดคอม86 เช่น ระบบปฏิบัติการที่ใช้ ข้อจำกัดเกี่ยวกับหน่วยความจำ เป็นต้น
- ศึกษาว่าในเวอร์ชวลแมชีนขนาดเล็ก (KVM) มีโครงสร้างและการทำงานอย่างไรบ้าง
- ศึกษาและทำความเข้าใจว่าแต่ละไฟล์ในเวอร์ชวลแมชีนขนาดเล็ก (KVM) ใช้ทำงานอะไร และมีความสัมพันธ์กันอย่างไร
- ศึกษาว่าในเวอร์ชวลแมชีนขนาดเล็ก (KVM) มีการจัดการหน่วยความจำอย่างไรบ้าง
- ศึกษาว่าในเวอร์ชวลแมชีนขนาดเล็ก (KVM) มีการจัดการเรื่องเวลาอย่างไรบ้าง
- ศึกษาว่าในเวอร์ชวลแมชีนขนาดเล็ก (KVM) มีการจัดการเรื่องโพรเซสอย่างไรบ้าง

- ศึกษาว่าในเวอร์ชวลแมชีนขนาดเล็ก (KVM) มีส่วนการทำงานที่ขึ้นกับระบบปฏิบัติการ (Platform-Dependent) อย่างไรบ้าง
- ทำการแก้ไขโค้ดในเวอร์ชวลแมชีนขนาดเล็ก (KVM) ที่ใช้ส่วนที่ขึ้นกับระบบปฏิบัติการ(Platform-Dependent) ให้สามารถทำงานบนระบบปฏิบัติการดอสได้
- ทำการแก้ไขโค้ดส่วนอื่นๆ ในเวอร์ชวลแมชีนขนาดเล็ก (KVM) เดิม ให้สามารถทำงานบนระบบปฏิบัติการดอสได้
- ทำการรีคอมไพล์โค้ดที่แก้ไขใหม่ทั้งหมด ทำการพัฒนาเป็นรอบ และจัดทำเป็นเวอร์ชั่นต่างๆ
- ทดสอบการทำงานของเวอร์ชวลแมชีนขนาดเล็กหรือเควีเอ็ม (KVM) ใหม่ที่ได้
- จัดทำเอกสารโครงงาน

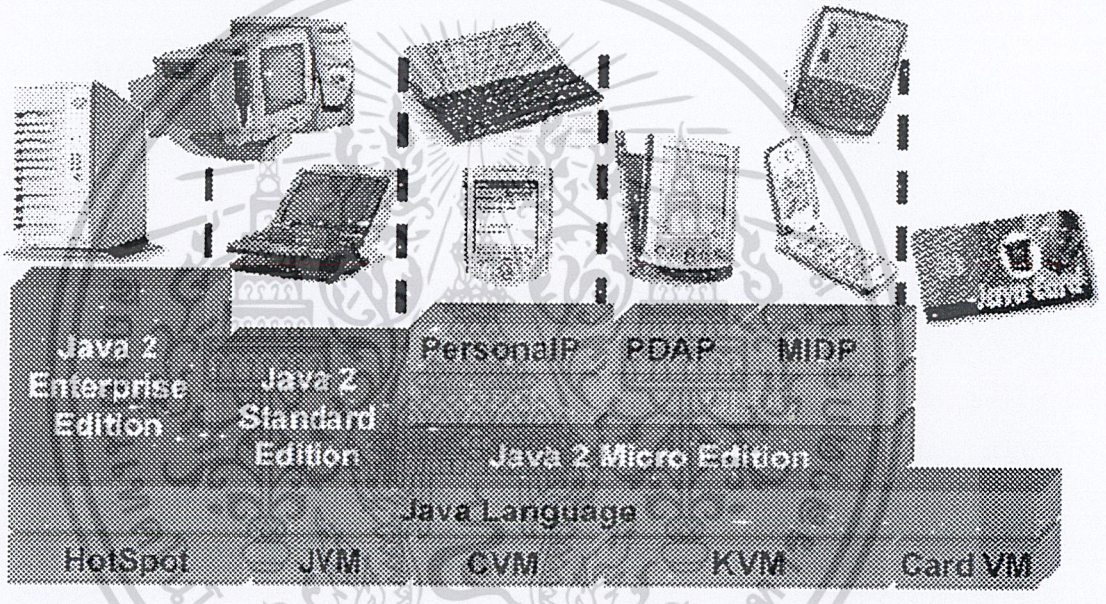
บทที่ 2
เทคโนโลยีที่เกี่ยวข้อง

สำหรับโครงงานนี้ จะมีเทคโนโลยีที่เกี่ยวข้องอยู่ 2 เรื่องด้วยกัน ได้แก่

2.1 จาวาเทคโนโลยี (Java™ Technology)

ภาษาจาวานั้น เป็นภาษาคอมพิวเตอร์ของบริษัท ซันไมโครซิสเต็มส์ ที่ได้รับการยอมรับโดยทั่วไปถึงประสิทธิภาพและประโยชน์ในการใช้งาน ซึ่งจาวานั้น มีจุดเด่นอยู่ที่ความง่ายในการเขียนโดยใช้การเขียนแบบเชิงวัตถุ (Object-Oriented Programing) ซึ่งง่ายต่อความเข้าใจและสามารถทำงานได้บนเครื่องในทุกๆ แพลตฟอร์ม

จาวาแบ่งโครงสร้างของภาษาออกเป็น 3 กลุ่ม โดยจะดูจากประสิทธิภาพ การใช้งาน ขนาดหน่วยความจำและอุปกรณ์ที่ใช้งาน ดังรูป



รูปที่ 2-1 จาวาเทคโนโลยี

- **J2ME (Java 2 Micro Edition)**

หมายถึงรูปแบบสภาวะการทำงานของโปรแกรมที่สร้างจากภาษาจาวา แต่เหมาะสมสำหรับการใช้งานบนอุปกรณ์อิเล็กทรอนิกส์ขนาดเล็ก เช่น โทรศัพท์มือถือ พีดีเอ เครื่องใช้ไฟฟ้า หรืออุปกรณ์ไร้สาย เป็นต้น

- **J2SE (Java 2 Standard Edition)**

หมายถึงรูปแบบสภาวะการทำงานของโปรแกรมที่สร้างจากภาษาจาวา ที่ทำงานบนคอมพิวเตอร์ทั่วไปในลักษณะสแตนอะโลน (Stand Alone) ประกอบด้วย

- Java Application คือลักษณะการทำงานในแบบทั่วไป สามารถติดตั้งและทำงานในทุกเครื่องที่ถูกติดตั้งจาวาเวอร์ชวลแมชีน

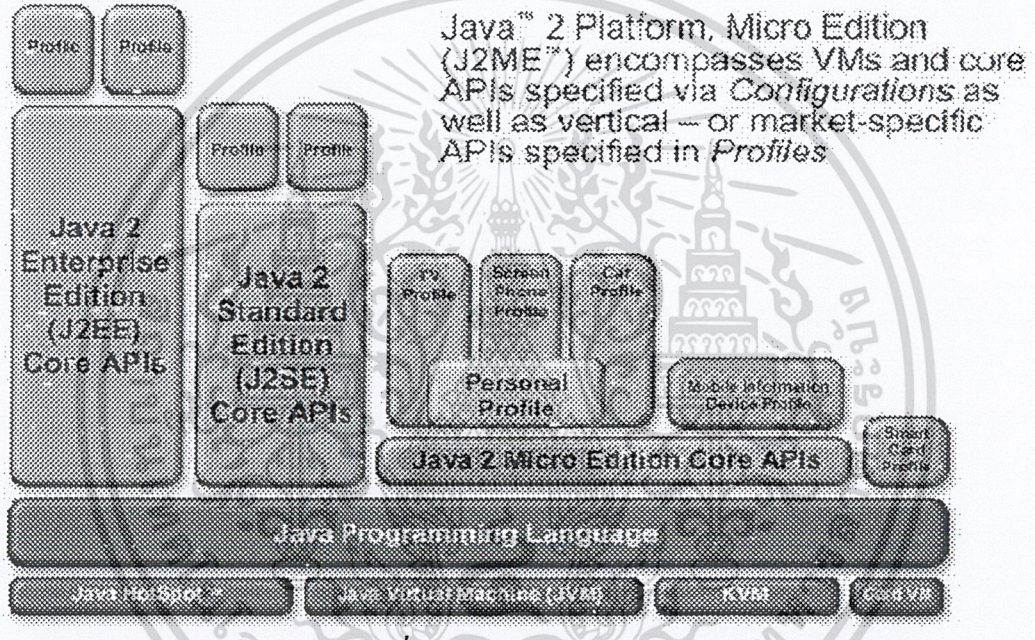
- Java Applet คือลักษณะการทำงานของโปรแกรมที่ต้องทำงานบนพื้นที่ใช้งานบราวเซอร์อีกที ดังนั้นบราวเซอร์จะต้องติดตั้งจาวาเวอร์ชวลแมชีนไว้รองรับการทำงาน

- JavaBean คือรูปแบบของชิ้นส่วนโปรแกรม ที่สร้างขึ้นมาเพื่อนำไปใช้พัฒนาโปรแกรม หรือใช้เพื่อทำงานร่วมภายในโปรแกรมอื่นๆ อีกทีหนึ่ง

- **J2EE (Java 2 Enterprise Edition)**

หมายถึงรูปแบบสภาวะการทำงานของโปรแกรมที่สร้างจากภาษาจาวา ที่ทำงานบนคอมพิวเตอร์ระดับเซิร์ฟเวอร์ เพื่อรองรับการใช้งานจากผู้ใช้จำนวนมากๆ ประกอบด้วย

- Java Servlet คือโปรแกรมสร้างจากภาษาจาวา ตามรูปแบบการสร้างเซิร์ฟเล็ต การใช้งานต้องติดตั้งบนเว็บเซิร์ฟเวอร์ที่สนับสนุนการทำงาน เมื่อเซิร์ฟเล็ตถูกเรียกใช้จากบราวเซอร์ เซิร์ฟเล็ตจะทำงานโดยอาศัยจาวาเวอร์ชวลแมชีน บนเครื่องเว็บเซิร์ฟเวอร์ และประมวลผลการทำงานให้ได้ข้อมูลเพื่อจัดส่งไปให้ผู้เรียกใช้จากบราวเซอร์ต่อไป เซิร์ฟเล็ตถูกเรียกใช้จากผู้ใช้ผ่านทางโพรโตคอลเอชทีทีพี และส่งผลลัพธ์เป็นข้อมูลเว็บไปยัง บราวเซอร์ของผู้ใช้อีกที

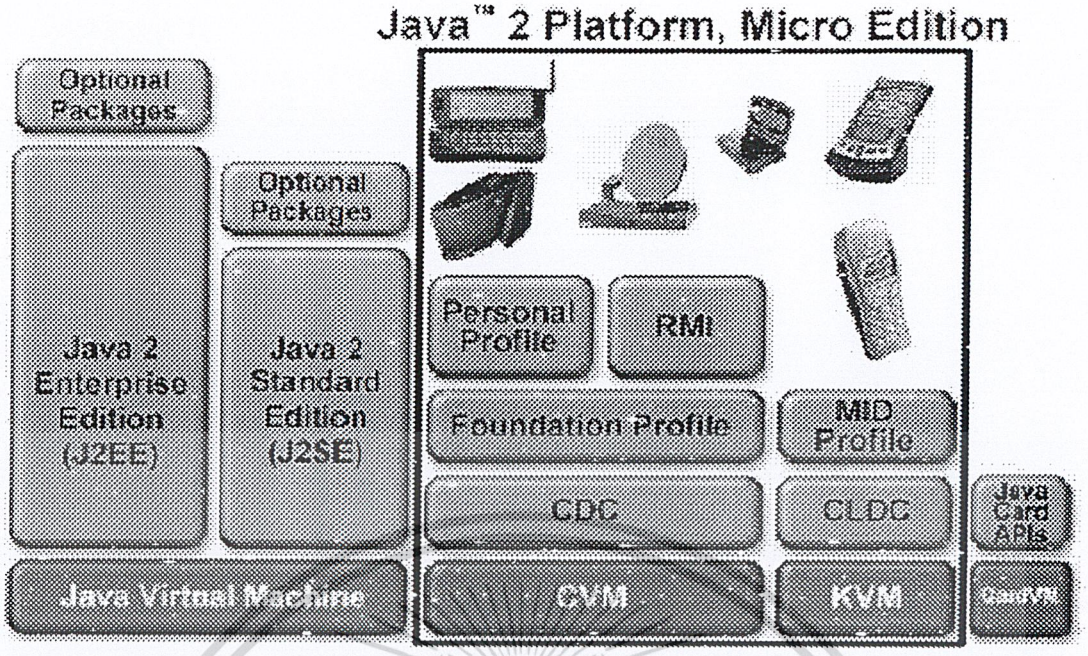


รูปที่ 2-2 โครงสร้างของจาวาทูแพลตฟอร์ม

2.1.1 จาวาทูไมโครเอดิชัน (J2ME™ Technology)

การพัฒนาโปรแกรมสำหรับอุปกรณ์ที่ทำงานในแบบไร้สายเช่น โทรศัพท์มือถือ เพจเจอร์สองทาง เป็นหัวข้อที่น่าสนใจในขณะนี้สำหรับนักพัฒนาโปรแกรม เนื่องจากข้อจำกัดของจอภาพ (Screen) หน่วยความจำ และการประมวลผลบนอุปกรณ์เหล่านั้น ทำให้การทำงานของโปรแกรมที่จะทำงานได้ ต้องมีคุณสมบัติพิเศษ ด้วยเทคโนโลยี จาวาทูไมโครเอดิชันจะสามารถนักพัฒนา สามารถสร้างโปรแกรมที่ทำงานบนอุปกรณ์ไร้สายต่างๆได้ เป็นอย่างดี

จาวาทูไมโครเอดิชัน (Java 2 Micro Edition) ถูกนำมาใช้ในอุปกรณ์ที่มีข้อจำกัดในด้าน การแสดงผล หน่วยความจำ ระบบประมวลผล การป้อนข้อมูล ซึ่งมีความแตกต่างจากการทำงานบนเครื่องคอมพิวเตอร์พีซีโดยทั่วไป ที่มีการพัฒนาฮาร์ดแวร์ที่มีความสามารถหลากหลายและรวดเร็ว การเรียนรู้จาวาทูไมโครเอดิชัน เป็นเรื่องง่ายสำหรับผู้ที่คุ้นเคยภาษาจาวามาแล้ว เรียกง่ายๆก็คือการทำความรู้จัก คลาสต่างๆ ที่มีให้ใช้ และสภาวะแวดล้อมที่จะใช้งานเท่านั้นว่าเป็นอย่างไร ซึ่งจำนวนคลาสต่างๆ ที่มีให้ใช้ ก็มีจำนวนเล็กน้อยเท่านั้น เมื่อเปรียบเทียบกับเอดิชันอื่นๆ

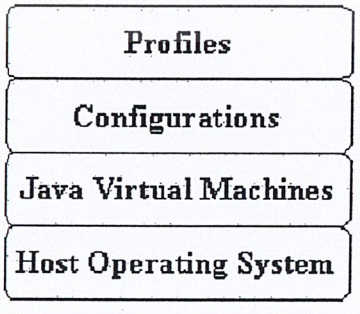


รูปที่ 2-3 จาวาทูไมโครเอดิชัน

2.1.2 โครงสร้างของจาวาทูไมโครเอดิชัน (J2ME Architectures)

โครงสร้างของจาวาทูไมโครเอดิชันประกอบด้วย 4 องค์ประกอบหลัก คือ

1. ระบบปฏิบัติการ (Operating System)
2. จาวาเวอร์ชวลแมชีน มี 2 แบบด้วยกัน คือ
   - เวอร์ชวลแมชีนขนาดเล็ก หรือ เควีเอ็ม (KVM)
   - ซีเวอร์ชวลแมชีน หรือ ซีวีเอ็ม (CVM)
3. คอนฟิกูเรชัน มี 2 แบบด้วยกัน คือ
   - ซีแอลดีซี (CLDC)
   - ซีดีซี (CDC)
4. โปรไฟล์ (Profile) เช่น โปรไฟล์สำหรับโทรศัพท์มือถือ (MIDP Profile) พีดีเอโปรไฟล์ (PDA Profile) เป็นต้น



รูปที่ 2-4 โครงสร้างของจาวาทูไมโครเอดิชัน

2.1.3 คอนฟิกูเรชัน (Configurations)

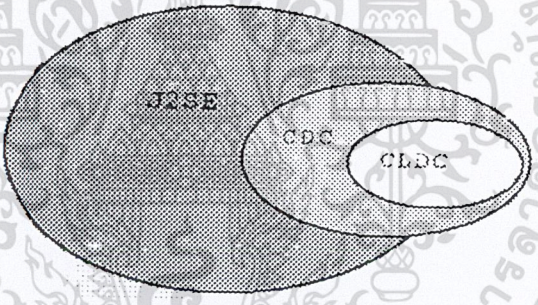
คอนฟิกูเรชัน คือข้อกำหนดสภาวะแวดล้อมในการทำงาน เป็นกลุ่มของคลาสไลบรารี่ ซึ่งเป็นเบสิคจาวาเอพีไอ (Java APIs) ซึ่งประกอบด้วย 3 ส่วนย่อย คือ

- เวอร์ชวลแมชชีน (VM) เพื่อเรียกใช้งานไบต์โค้ด ที่นักพัฒนาสร้างขึ้นมา
- เนทีฟโค้ด (Native Code) ซึ่งจะติดต่อกับระบบ และชิ้นส่วนต่างๆ ของอุปกรณ์
- Runtime Classes คือกลุ่มคลาสที่ถูกสร้างขึ้นไว้ รองรับการทำงานเมื่อใช้งานแอพพลิเคชั่น J2ME

ในการใช้งานคอนฟิกูเรชัน อุปกรณ์ใดๆต้องสามารถทำงานกับข้อกำหนดมาตรฐานที่กำหนดไว้ และต้องสามารถใช้งานรันไทม์คลาส ตามข้อกำหนดได้ ซึ่งคลาสดังกล่าวมีขนาดเล็กมาก คลาสเหล่านี้ทำงานติดต่อกับระบบเท่านั้น ยังไม่มีส่วนของการทำงานในส่วนอินเตอร์เฟสกับผู้ใช้

ในจาวาทูไมโครเอดิชัน มีคอนฟิกูเรชัน 2 แบบ คือ

1. CLDC : Connected Limited Device Configuration
2. CDC : Connected Device Configuration



รูปที่ 2-5 ขอบเขตของคอนฟิกูเรชัน

2.1.3.1 CLDC (Connected Limited Device Configuration)

เป็นข้อกำหนดสำหรับอุปกรณ์ที่มีข้อจำกัดมากๆ เนื่องจากมีหน่วยความจำน้อย และโปรเซสเซอร์ที่ทำงานไม่เร็วนัก ซึ่งในการทำงานของเวอร์ชวลแมชีน ในรูปแบบนี้ไม่ได้ทำงานในส่วนไฟนอลไลเซชั่น (finalization) ของคลาส และขนาดของรันไทม์คลาส ยังมีขนาดที่เล็กและจำนวนน้อยด้วย รูปแบบจะเป็นคลาสไลบรารี่บางส่วนของจาวา และเป็นคอนฟิกูเรชันสำหรับอุปกรณ์ไร้สายขนาดเล็ก เช่น โทรศัพท์มือถือ โดยที่อุปกรณ์นั้นจะต้องมี หน่วยความจำ 60-512 กิโลไบต์ คอนฟิกูเรชันนี้จะทำงานร่วมกันกับเวอร์ชวลแมชีนขนาดเล็ก โดยใช้โพรเซสเซอร์ขนาด 16 บิต

การทำงานภายในคอนฟิกูเรชันนี้ ประกอบไปด้วย

1. โครงสร้างภาษาจาวาและลักษณะของเวอร์ชวลแมชีน
2. จาวาไลบรารี่หลัก

3. อินพุท/เอาท์พุท

4. ระบบเครือข่าย (Networking)

5. ระบบความปลอดภัย (Security)

6. ความเป็นสากล (Internationalization)

สิ่งที่ไม่อยู่ในคอนฟิกูเรชัน แต่อยู่ในโปรไฟล์แทน คือ

1. การจัดการเกี่ยวกับไลฟ์ไซเคิล

2. ส่วนติดต่อกับผู้ใช้ (User interface)

3. การจัดการอีเวนท์ต่างๆ (Event handing)

4. รูปแบบการสร้างแอพพลิเคชั่นระดับสูง (High-level Application Model)

คอนฟิกูเรชันแบบซีแอลดีซีมีแพ็คเกจต่างๆ ดังนี้

1. บางส่วนของจาวาทูสแตนดาดเอดิชัน (Standard Edition)

- java.lang
- java.io
- java.util

2. ส่วนที่มีเฉพาะในคอนฟิกูเรชันแบบซีแอลดีซี

- javax.microedition.io

CLDC Libraries: java.lang.*

- Object
- Runtime
- Thread
- String
- Throwable
- Boolean
- Short
- Long
- Class
- System
- Runnable
- StringBuffer
- Math
- Byte
- Integer
- Character

CLDC Libraries : java.io.*

- InputStream
- DataInput
- Reader
- ByteArrayInputStream
- DataInputStream
- InputStreamReader
- OutputStream
- DataOutput
- Writer
- ByteArrayOutputStream
- DataOutputStream
- OutputStream Writer
- PrintStream

CLDC Libraries : java.util.*

- Calendar
- Date
- TimeZone
- Vector
- Stack
- Hashtable
- Enumeration
- Random

2.1.3.2 CDC (Connected Device Configuration)

ในขณะที่คอนฟิกูเรชันแบบนี้ บรรจุเอา เวอร์ชวลแมชีนที่ทำงานได้เต็มรูปแบบ และบรรจุเอารันไทม์คลาสที่มีจำนวนมากรองรับการใช้งานได้ยืดหยุ่นกว่า แน่นอนว่า ในการทำงานจะต้องอาศัยสภาวะของทรัพยากร เช่น หน่วยความจำ และโปรเซสเซอร์ที่ทำงานเร็วมากกว่าด้วย เป็นคอนฟิกูเรชันที่เหมาะสำหรับอุปกรณ์ที่มีความสามารถสูงมากกว่าอุปกรณ์แบบซีแอลดีซี เช่น พีดีเอ คอมมูนิเคเตอร์ เป็นต้น โดยที่คอนฟิกูเรชันแบบนี้จะทำงานร่วมกับซีเวอร์ชวลแมชีน

คอนฟิกูเรชันแบบซีแอลดีซีมีแพ็คเกจต่างๆ ดังนี้

- java.lang : เป็นส่วนคลาสหลักของจาวา
- java.util : เป็นส่วนสนับสนุนจาวา
- java.net : จัดการเกี่ยวกับการส่งข้อมูลแบบยูดีพี (UDP Datagram) และไฟล์อินพุท/เอาท์พุท (File I/O)
- java.text : รองรับแบบ I18n
- java.security : จัดการเรื่องความปลอดภัย
- java.io : จัดการเกี่ยวกับจาวาไฟล์อินพุท/เอาท์พุท (Java File I/O)

2.1.4 โปรไฟล์ (Profiles)

โปรไฟล์ เป็นการเพิ่มเติมคลาสลงในคอนฟิกูเรชัน เพื่อเพิ่มเมธอดพิเศษให้ทำงานกับอุปกรณ์ซึ่งรวมถึงคลาสและเมธอดที่รับผิดชอบในการสร้างส่วนติดต่อกับผู้ใช้ด้วย การใช้งานโปรไฟล์ อุปกรณ์ต่างๆ ต้องเป็นอุปกรณ์ที่ผลิตขึ้นเอง สามารถทำงานได้ตามข้อกำหนดรายละเอียดของกลุ่ม JCP โปรไฟล์ เป็นลักษณะเฉพาะของคลาสไลบรารี่ของจาวา ซึ่งจะเก็บข้อมูลที่ต้องใช้โดยแบ่งออกไปตามชนิดของอุปกรณ์ต่างๆ ในปัจจุบันมีโปรไฟล์ เกิดขึ้นมาหลากหลายรูปแบบ ดังนี้

- **Mobile Information Device Profile (MIDP)**

ทำงานบนซีแอลดีซี สำหรับสร้างแอพพลิเคชั่นที่ทำงาน บนโทรศัพท์มือถือ และเพจเจอร์แบบอินเตอร์เอ็คทีฟ หรืออุปกรณ์ที่สนับสนุนการใช้งานโพรโตคอลเอชทีทีพี ประกอบด้วย

- javax.microedition.midlet
- javax.microedition.lcdui
- javax.microedition.rms

- PDA Profile

เป็นโปรไฟล์สำหรับพีดีเอ ที่เพิ่มขยายความสามารถของโปรไฟล์ของมือถือ ใช้สำหรับอุปกรณ์ที่มีทรัพยากรในการทำงานสูง และทำงานบนซีดีซี

- Foundation Profile

ทำงานบนซีดีซี เพิ่มเติมคลาสในการใช้งานและทำงานในรูปแบบที่ใช้กับจาวาสแตนดาดเอดิชัน แต่ไม่มีส่วนที่เป็นการติดต่อกับผู้ใช้

- Personal Profile

สร้างมาจากฟาวเดชั่นโปรไฟล์ (Foundation Profile) มีส่วนที่เป็นการติดต่อกับผู้ใช้เป็นแบบเอดับเบิลยูที (AWT) และจาวาแอพเพล็ต

- RMI Profile

สร้างมาจากฟาวเดชั่นโปรไฟล์ (Foundation Profile) มีอาร์เอ็มไอสำหรับใช้งานในอุปกรณ์ต่าง ๆ ได้

2.1.5 จาวาเวอร์ชวลแมชีน (Java Virtual Machine)

ใน ภาษาจาวา เวอร์ชวลแมชีนจะอยู่ระหว่างระบบปฏิบัติการกับคอนฟิกูเรชันซึ่งจะทำหน้าที่ในการแปลงไบต์โค้ด ของโปรแกรมให้เป็นโค้ดที่เครื่องสามารถเข้าใจได้ (Machine-Level Code) และจะทำการจัดการกับสภาวะ (environment) ต่างๆ ของระบบ เช่น จัดการเกี่ยวกับหน่วยความจำ (System memory) เธรด (Tread) เป็นต้น ใน0k;k ไมโครเอดิชันนี้จะมีเวอร์ชวลแมชีน 2 แบบด้วยกัน คือ

2.1.5.1 เวอร์ชวลแมชีนขนาดเล็ก หรือ เควีเอ็ม (KVM)

- KVM ย่อมาจาก Kilo Virtual Machine
- เป็นเวอร์ชวลแมชีนสำหรับอุปกรณ์ขนาดเล็ก เช่น โทรศัพท์มือถือ ซึ่งจะใช้หน่วยความจำทั้งหมดแค่เพียง 128 kB ในการรันแอพพลิเคชั่น
- ขนาดของเวอร์ชวลแมชีนขณะทำงานแค่ 40-80 kB
- ใช้ภาษา C สร้าง
- ใช้โพรเซสเซอร์ขนาด 16-32 bit

ข้อจำกัดของเวอร์ชวลแมชีนขนาดเล็ก

- ไม่สนับสนุนการทำงานแบบโฟลทติ้งพ้อยท์ (floating-point)
- ไม่สนับสนุนการทำงานของจาวาเนทีฟอินเตอร์เฟส (Java Native Interface : JNI)
- ไม่สนับสนุนการโหลดคลาสแบบที่ผู้ใช้กำหนดเอง (user-defined class loaders)
- ไม่มีการทำเธรดกรุ๊ป (hread groups)

- ไม่มีการทำไฟนอลไลเซชั่น (finalization)
- จำกัดการทำส่วนการจัดการเกี่ยวกับข้อผิดพลาด

โครงสร้างหลักของเวอร์ชวลแมชีนขนาดเล็ก

ในเวอร์ชวลแมชีนขนาดเล็กประกอบไปด้วย ไ ดเรคทอรี่ต่างๆ โดยจะมีส่วนที่เก็บโค้ดโปรแกรมที่เป็นไฟล์นามสกุล ". c" (source code) อยู่ในไ ดเรคทอรี่ src และไฟล์เฮดเดอร์ที่ใช้ (include file) เก็บอยู่ในไ ดเรคทอรี่ h เราสามารถแบ่ง เวอร์ชวลแมชีนขนาดเล็กตามไ ดเรคทอรี่และการทำงานดังนี้

1. VmCommon : เป็นโค้ดของเวอร์ชวลแมชีนขนาดเล็กในส่วนที่ไม่ขึ้นอยู่กับแพลตฟอร์ม (Platform-Independent) ไฟล์ใน VmCommon นั้นจะเป็นไฟล์ที่ทำงานเกี่ยวกับหน้าที่การทำงานหลักๆ ของเวอร์ชวลแมชชีน เช่น StartJVM class events field frame garbage global cache interpret hashtable jar loader log native pool runtime thread verifier เป็นต้น

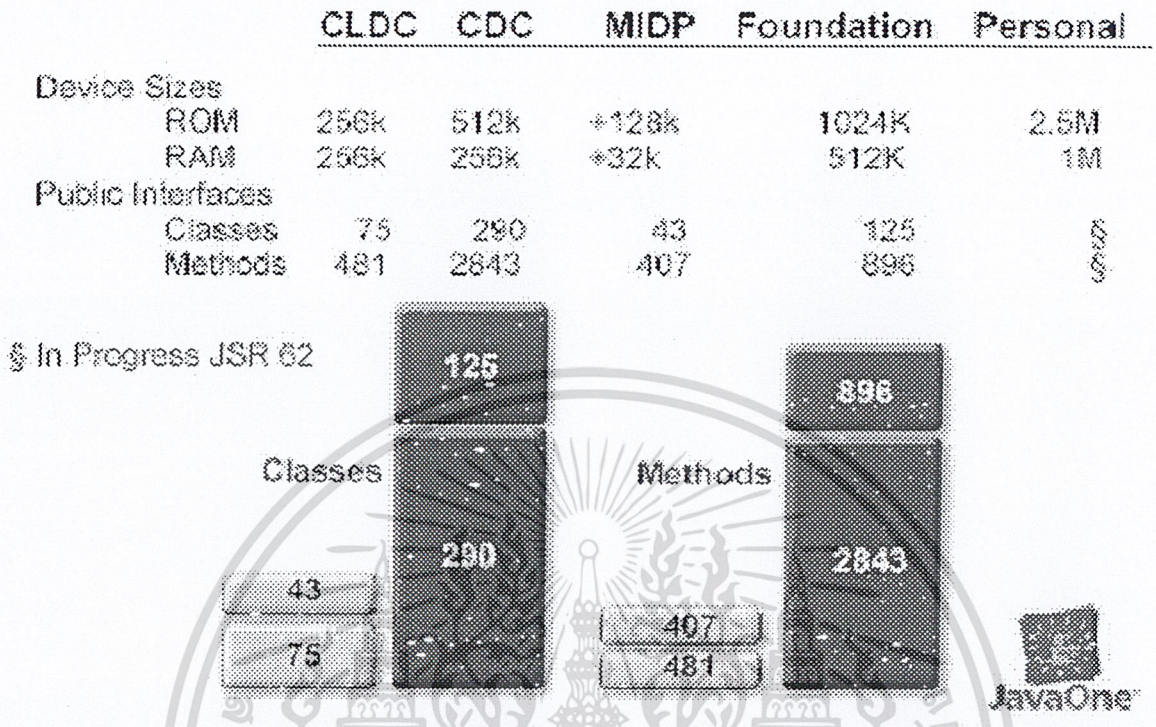
2. VmExtra : เป็นโค้ดของเวอร์ชวลแมชีนขนาดเล็กในส่วนการทำงานเกี่ยวกับระบบเครือข่าย และคอมโพเนนท์เพิ่มเติมต่างๆ เช่น network networkPrim ที่ทำงานเกี่ยวกับเครือข่าย เป็นต้น

3. VmPort : เป็นโค้ดของเวอร์ชวลแมชีนขนาดเล็กในส่วนการทำงานเกี่ยวกับระบบปฏิบัติการที่จะนำเวอร์ชวลแมชีนขนาดเล็กไปติดตั้ง เป็นไฟล์เกี่ยวกับลักษณะพิเศษในการเชื่อมต่อการทำงานเข้ากับระบบปฏิบัติการนั้นๆ เช่น VmWin สำหรับวินโดว์ ก็จะมีไฟล์เกี่ยวกับการจัดการทางด้านกราฟฟิกของวินโดว์ เป็นต้น ซึ่งจะเรียกชื่อส่วนนี้เปลี่ยนไปตามระบบปฏิบัติการที่ใช้ เช่น VmWin สำหรับวินโดว์ VmUnix สำหรับยูนิก เป็นต้น

2.1.5.2 ซีเวอร์ชวลแมชีน หรือ ซีวีเอ็ม (CVM)

- CVM ย่อมาจาก Compact Virtual Machine
- มีขนาดใหญ่กว่าเวอร์ชวลแมชีนขนาดเล็กและมีคลาส ต่างๆ ประมาณ 40% ของจาวาสแตนดาดเอดิชัน
- ใช้หน่วยความจำ 2-16 เมกะไบต์
- ใช้โพรเซสเซอร์ขนาด 32 บิต
- ถูกสร้างโดยใช้ภาษาซี และ แอสเซมบลี

2.1.6 ขนาดของจาวาไมโครเอดิชันในการนำไปติดตั้ง (Sizes of Public Interfaces)

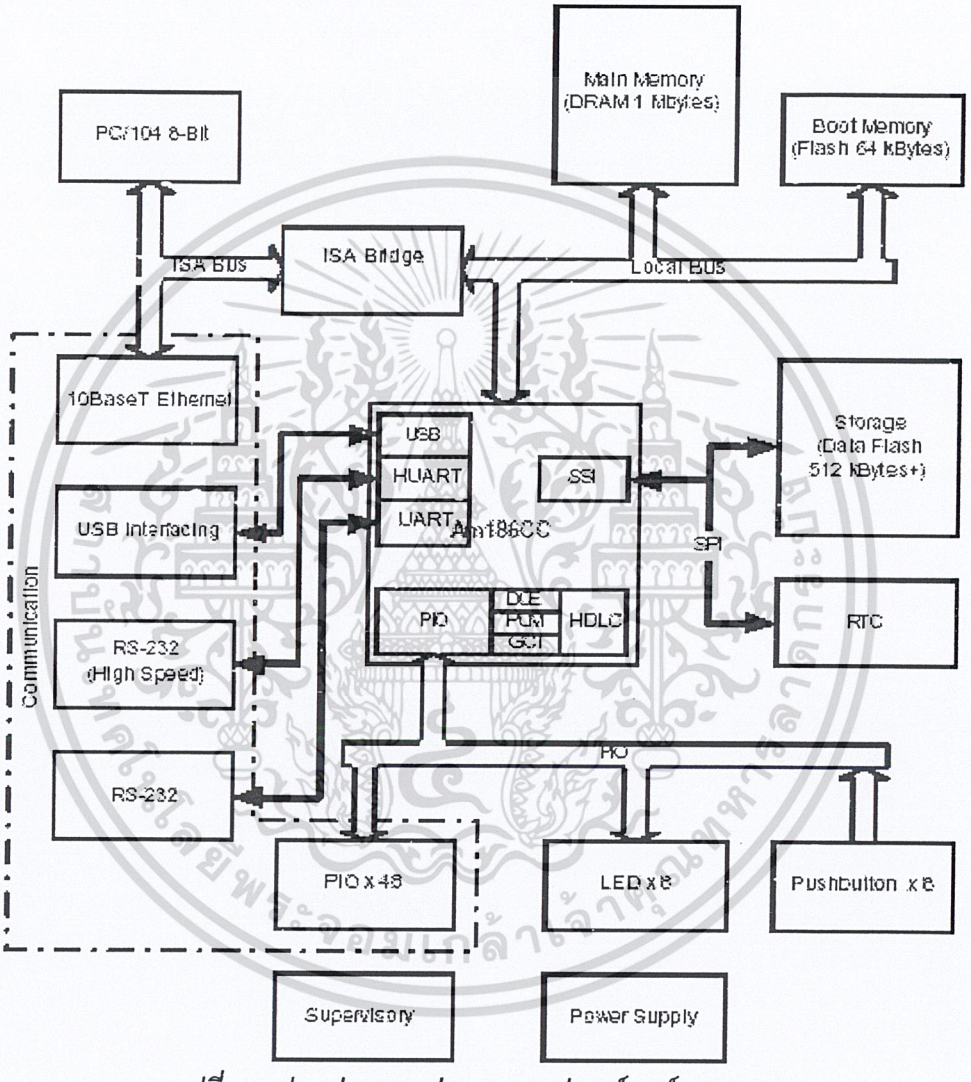


รูปที่ 2-6 ขนาดของจาวาไมโครเอดิชันในการนำไปติดตั้ง (Sizes of Public Interfaces)

2.2 อุปกรณ์บอร์ดคอม86

อุปกรณ์บอร์ดคอม86 (Com86 board) เป็นบอร์ดที่อยู่ในชุดพัฒนาคอม86 (Com86 platform) ซึ่งเป็นแพลตฟอร์มสำหรับการพัฒนาอุปกรณ์ทางด้านระบบฝังตัวโดยใช้ไมโครคอนโทลเลอร์ขนาด 16 บิต ออกแบบและพัฒนาโดยห้องปฏิบัติการระบบควบคุมด้วยสมองกล เพื่อเป็นการสนับสนุนการเรียนรู้และพัฒนาอุปกรณ์ทางด้านระบบฝังตัวของประเทศไทย บอร์ดคอม86 มีส่วนประกอบต่างๆดังนี้

2.2.1 สถาปัตยกรรมทางด้านฮาร์ดแวร์



รูปที่ 2-7 ส่วนประกอบต่าง ๆของอุปกรณ์บอร์ดคอม86

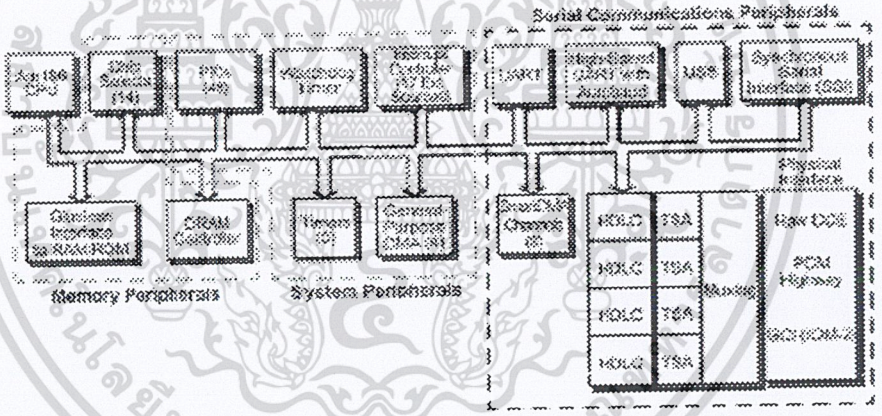
ไมโครโพรเซสเซอร์

หน่วยประมวลผลกลางของบอร์ดคอม86 ได้ออกแบบโดยใช้ไมโครคอนโทรลเลอร์ขนาด16 บิตในตระกูล E86 ของบริษัทเอเอ็มดีชื่อว่า Am186CC ซึ่งเป็นคอมมิวนิเคชันไมโครคอนโทรลเลอร์ (Communication microcontroller) ที่ใช้แกนหลัก (core) ของ 80186 โดยได้เพิ่มความสามารถทางด้านการสื่อสารและส่วนประกอบอื่นๆ โดยมีส่วนสำคัญที่ได้เพิ่มเข้าไปดังนี้

- ส่วน HDLC (High–level Data Link Control) สำหรับเป็นส่วนจัดการเรื่องการติดต่อสื่อสาร

แบบHDLCที่สามารถเชื่อมต่อกับ DCE, PCM Highway และ GCI (IOM-2) ผ่านทาง PIO โดยมี TSAs (Time Slot Assigners) สำหรับการทำงานในแบบมัลติเพลกซ์ (Multiplex)

- USB Peripheral Controller สำหรับการเชื่อมต่อให้เป็นอุปกรณ์แบบ USB
- HUART (High-speed UART) สำหรับใช้งานในการเชื่อมต่อแบบอนุกรมความเร็วสูง โดยสามารถสื่อสารได้ในอัตราเร็วสูงสุด 460 Kbit /s สามารถตรวจสอบอัตราเร็วในการเชื่อมต่อได้อัตโนมัติ (Automatic Baud Rate Detection) และมีฮาร์ดแวร์ Handshaking
- UART (Universal Asynchronous Receiver/Transmitter) สำหรับการเชื่อมต่อแบบอนุกรม โดยสามารถสื่อสารได้ในอัตราเร็วสูงสุด 115.2 Kbit/s และมีฮาร์ดแวร์ Handshaking
- SSI (Synchronous Serial Interface) สำหรับการเชื่อมต่อสื่อสารแบบอนุกรมความเร็วสูง (25 Mbit/s) ซึ่งสามารถสื่อสารแบบ SPI ได้
- Programmable timer ขนาด 16 บิต 3 ตัว
- Hardware watchdog timer
- ดีแรมคอนโทรลเลอร์ (DRAM Controller)
- อินเตอร์รัพท์คอนโทรลเลอร์ (Interrupt controller) 36 มาสก์เอเบิลอินเตอร์รัพท์



รูปที่ 2-8 บล็อกไดอะแกรมแสดงองค์ประกอบของไมโครคอนโทรลเลอร์ AM186CC

ไมโครคอนโทรลเลอร์ Am186CC นี้เป็นไมโครคอนโทรลเลอร์ที่ที่มีสถาปัตยกรรมแบบ X86 มีชุดคำสั่งที่เข้ากันได้กับชุดคำสั่งของ 80286 (Real Mode) สนับสนุนการอ้างอิงหน่วยความจำหลักได้ถึง 1-เมกะไบต์และไมโครโพรเซสเซอร์ทำงานด้วยความเร็ว 12 หรือ 24 เมกะเฮิรตซ์

หน่วยความจำหลัก

Main memory

หน่วยความจำหลักของบอร์ดคอม 86 เป็นหน่วยความแบบดีแรม (DRAM) ขนาด 1เมกะไบต์เป็นหน่วยความจำสำหรับการทำงานของระบบปฏิบัติการและแอพพลิเคชัน

Boot Memory

เป็นหน่วยความจำแบบแฟลช (Flash) ที่สำหรับใช้เก็บ BIOS และโปรแกรมสำหรับเริ่มการทำงานของระบบ

Storage

บอร์ดคอม86 มีส่วนเก็บข้อมูลที่เป็นหน่วยความจำแบบนอนโวลาไทล์ (Non-Volatile) เป็นหน่วยความจำ แบบแฟลช (Flash memory) ในเบื้องต้นมีขนาด 512 กิโลไบต์ สามารถขยายได้โดยการเปลี่ยนหน่วยความจำ data flash บนบอร์ด

อุปกรณ์ต่อเชื่อม

RTC

เป็นส่วนของ Real-Time Clock มีหน้าที่สำหรับบอกวันและเวลาของระบบ

PIO

PIO (Programmable Input/Output) เป็นส่วนของการเชื่อมต่อกับอุปกรณ์ภายนอกเพื่อควบคุมหรืออ่าน สถานะต่างๆของอุปกรณ์โดยสามารถโปรแกรมได้

Push button

เป็นสวิตช์ชนิดกดติดปล่อยดับจำนวน 8 ตัว เชื่อมต่อกับ PIO สามารถเลือกใช้หรือไม่ใช้งานได้ตามต้องการ

LED

เป็นLED สำหรับใช้แสดงผลจำนวน 8 ดวง เชื่อมต่อกับ PIO สามารถเลือกใช้หรือไม่ใช้งานได้ตามต้องการ

Power supply

เป็นส่วนที่ทำการจ่ายกระแสไฟฟ้าในระดับแรงดันต่างๆให้แก่ระบบทั้งหมดโดยรับแรงดันจากภายนอก12 โวลต์

Supervisory

เป็นส่วนของการสร้างสัญญาณรีเซตระบบและตรวจสอบแรงดันของ พาวเวอร์ซัพพลาย

ISA Bridge

เป็นส่วนที่ทำการเชื่อมต่อ ISA บัสเข้ากับโลคอลบัส (Local Bus) โดยทำหน้าที่จัดการสัญญาณต่างๆใน การเชื่อมต่อสื่อสารกันระหว่างบัสทั้ง 2 ด้านให้มีสัญญาณตรงตามมาตรฐานของ ISA BUS

PC/104 8 bit Compatible connector

เป็นคอนเนคเตอร์สำหรับเชื่อมต่อในลักษณะเดียวกันกับมาตรฐานPC/104 แบบ 8 บิต

10BaseT Ethernet

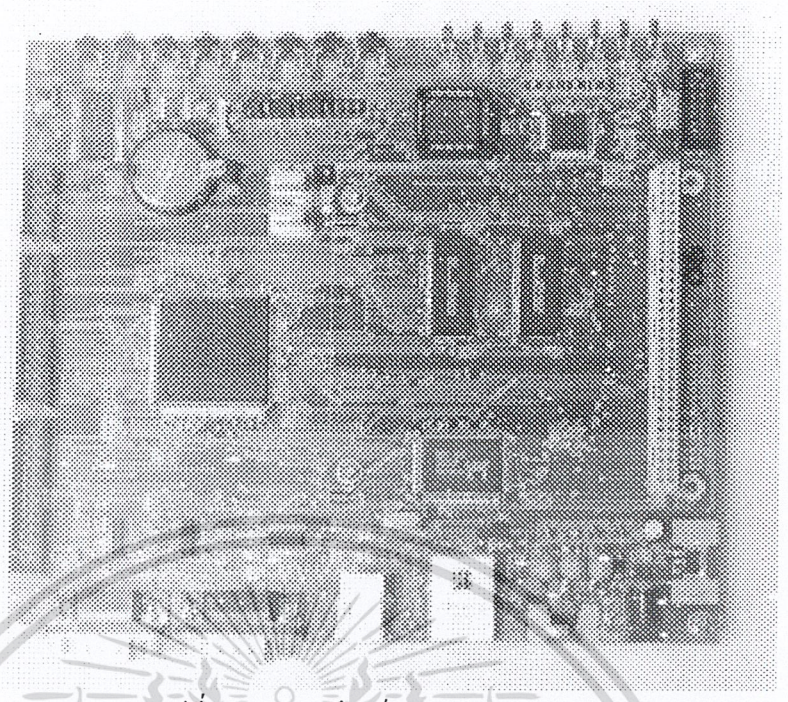
เป็นส่วนของการสื่อสารกับเครือข่าย Ethernet แบบ 10BaseT

USB Interface

เป็นส่วนเชื่อมต่อแบบ USB แบบ TYPE B

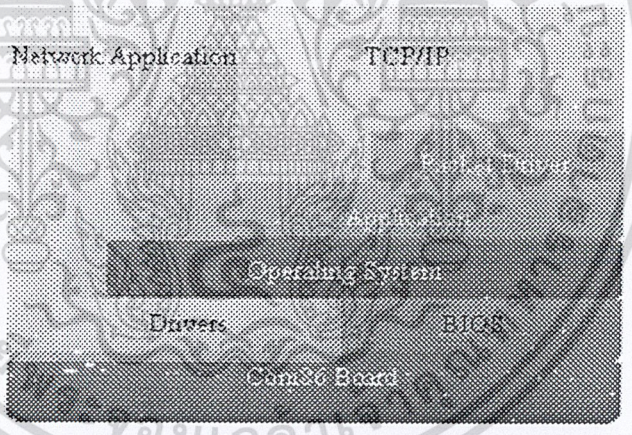
RS-232

เป็นพอร์ตสำหรับเชื่อมต่อกับภายนอกแบบ DB9 มีจำนวณ 2 พอร์ต



รูปที่ 2-9 อุปกรณ์บอร์ดคอม 86

2.2.2 สถาปัตยกรรมทางด้านซอฟต์แวร์



รูปที่ 2-10 สถาปัตยกรรมทางด้านซอฟท์แวร์บนอุปกรณ์บอร์ดคอม86

จากภาพแสดงถึงลำดับชั้นต่างๆของซอฟต์แวร์ที่ทำงานอยู่บนบอร์ด คอม86โดยแต่ละชั้นมีความหมายต่างๆ ดังนี้

BIOS

เป็นโปรแกรมที่คอยควบคุมการทำงานของฮาร์ดแวร์ต่างๆบนบอร์ดคอม86 เป็นส่วนที่จัดการการสื่อสารระหว่างฮาร์ดแวร์กับแอพพลิเคชันซอพท์แวร์ และระบบปฏิบัติการ

Driver

เป็นโปรแกรมขนาดเล็กที่ทำหน้าที่ ติดต่อสื่อสารกับฮาร์ดแวร์ อื่นๆที่ไบออสไม่ได้รองรับการทำงานไว้เช่น ฮาร์ดแวร์ที่สร้างขึ้นมาเพิ่มเติมเพื่อเชื่อมต่อเข้ากับบอร์ดคอม86

Operating System

ระบบปฏิบัติการเป็นโปรแกรมที่ควบคุมการทำงานของโปรแกรมต่างๆในระบบ ทั้งในส่วนของการจัดการหน่วยความจำ เป็นส่วนสื่อสารระหว่าง โปรแกรมแอพพลิเคชันของผู้ใช้และ ส่วนของฮาร์ดแวร์ ระบบต่างๆ

Packet Driver

เป็นโปรแกรมจัดการเกี่ยวกับแพ็กเก็ตของการส่งข้อมูล สำหรับแอพพลิเคชันทางด้านการสื่อสาร

TCP/IP

เป็นโปรแกรมจัดการทางด้าน โพรโตคอลทีซีพี/ไอพีสำหรับแอพลิเคชันทางด้านการสื่อสาร

Network Application

แอพพลิเคชั่นทางด้านการสื่อสาร เป็นโปรแกรมแอพพลิเคชันทางด้านการสื่อสารต่างๆ เช่น โปรแกรมเว็บเซิร์ฟเวอร์ในระบบฝังตัว (Embedded web server) เป็นต้น

Application

เป็นโปรแกรมแอพลิเคชันอื่นๆ ที่ไม่ได้ใช้ความสามารถทางด้านการสื่อสารเช่นโปรแกรมควบคุมการทำงานของหุ่นยนต์ หรือเครื่องจักร เป็นต้น

2.2.3 จุดเด่นของบอร์ด คอม86

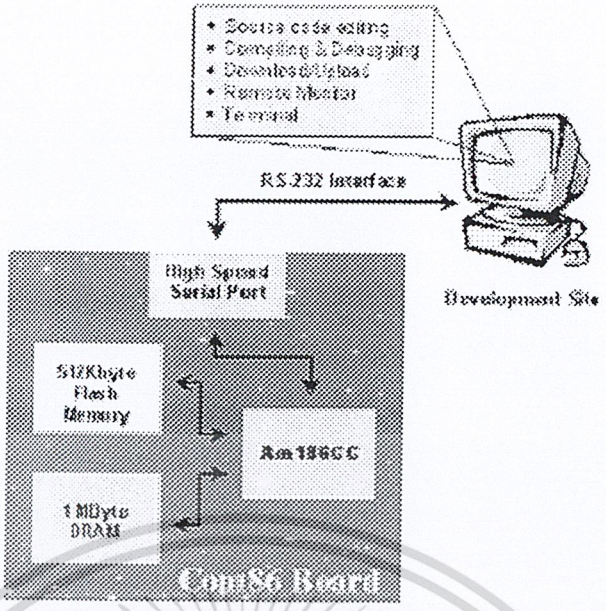
บอร์ดคอม86 มีคุณสมบัติเด่นสำหรับการพัฒนาอุปกรณ์ทางด้านระบบฝังตัว ดังนี้

2.2.3.1 เป็นไมโครคอนโทรลเลอร์ที่มีความสามารถสูง

บอร์ดคอม86 เลือกใช้ไมโครคอนโทรลเลอร์Am186 CC ของบริษัท AMD ซึ่งเป็นไมโครคอนโทรลเลอร์ขนาด 16 บิต สามารถทำงานได้ที่ ความเร็ว 25 40 และ 50 MHz ซึ่งทำให้มีความสามารถในการประมวลผลเพิ่มขึ้นมากเมื่อเทียบกับการพัฒนาโดยใช้ไมโครคอนโทรลเลอร์ขนาด 8 บิตที่นิยมใช้กันอยู่ในปัจจุบัน นอกจากนี้ยังได้รวมเอาอุปกรณ์พื้นฐานต่างๆ ที่จำเป็นเข้าไว้ในตัว เช่น DRAM controller, Interrupt controller, UART เป็นต้นซึ่งทำให้ง่ายต่อการออกแบบอุปกรณ์ เพื่อทำเป็นสินค้าหลังการพัฒนา นอกจากนี้ ยังเพิ่มส่วนของการสื่อสารต่างๆเข้าไป ทำให้มีความสามารถทาง
ด้านการสื่อสารมากขึ้น เหมาะกับการพัฒนาอุปกรณ์ที่ต้องใช้คุณสมบัติการสื่อสารต่างๆ เมื่อเทียบกับการใช้ไมโครคอนโทรลเลอร์ขนาดเล็กแบบเดิมซึ่งมีข้อจำกัดต่างๆอยู่มากมาย

2.2.3.2 ง่ายในการเรียนรู้และพัฒนา

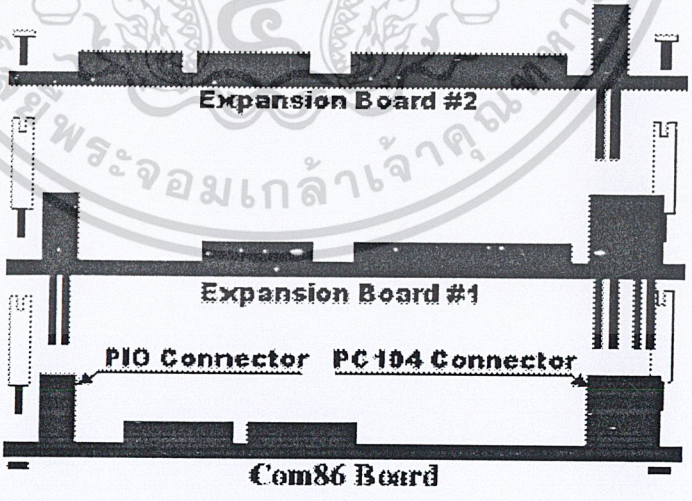
บอร์ดคอม86 มีสถาปัตยกรรมของไมโครคอนโทรลเลอร์แบบ x86 ซึ่งเป็นสถาปัตยกรรมเดียวกับคอมพิวเตอร์ส่วนบุคคลทั่วไป โดยมีชุดคำสั่งที่เข้ากันได้กับชุดคำสั่งของ 80286 (Real mode) ทำให้ง่ายต่อการเรียนรู้และพัฒนาโปรแกรม และยังสามารถพัฒนาและทดสอบซอฟต์แวร์ต่างๆได้บนคอมพิวเตอร์ ก่อนที่จะทดสอบกับบอร์ด คอม86 ทำให้สามารถพัฒนาอุปกรณ์ทางด้านระบบฝังตัวต่างๆได้อย่างรวดเร็ว



รูปที่ 2-11 รูปแบบการพัฒนาโปรแกรมโดยใช้อุปกรณ์บอร์ดคอม86

2.2.3.3 มีความสามารถในการเพิ่มขยายได้

ในส่วนของการเพิ่มขยายของฮาร์ดแวร์สามารถทำได้โดยผ่านทางคอนเนคเตอร์ ของบอร์ดที่ออกแบบไว้ตามมาตรฐานPC 104 ซึ่งจะมีการเชื่อมต่อคล้ายกับมาตรฐาน ISA BUS ของ คอมพิวเตอร์ส่วนบุคคล ซึ่งผู้พัฒนาสามารถออกแบบฮาร์ดแวร์ให้เป็นลักษณะของบอร์ดขยายและนำมาเชื่อมต่อกับบอร์ดคอม86ได้ทันที และเขียนซอฟต์แวร์ไดร์เวอร์สำหรับควบคุมส่วนส่วนเพิ่มเข้ามาใหม่ซึ่งจะทำงานร่วมกับระบบปฏิบัติการในการรองรับการทำงานของแอพพลิเคชันต่างๆ



รูปที่ 2-12 เชื่อมต่อของบอร์ดขยายกับอุปกรณ์บอร์ดคอม86

2.2.3.4 มีเครื่องมือในการพัฒนาโปรแกรมมาก

เนื่องจากในการพัฒนาโปรแกรมเป็นเสมือนกับการพัฒนาบนสถาปัตยกรรม X86 ของคอมพิวเตอร์ ส่วนบุคคลทั่วไป ดังนั้นจึงสามารถนำโปรแกรมเครื่องมือต่างๆที่ใช้พัฒนาโปรแกรมสำหรับคอมพิวเตอร์ส่วนบุคคลมาใช้

งานได้มากมายและเป็นที่คุ้นเคยของนักพัฒนาทำให้สามารถพัฒนาอุปกรณ์ได้อย่างรวดร็ว ลดต้นทุนในการพัฒนาต่างๆได้เป็นจำนวนมาก

2.2.4 แอพพลิเคชันเป้าหมายของชุดพัฒนา (Target Application)

จากความสามารถของไมโครคอนโทรลเลอร์ของบอร์ดคอม86 ทำให้การประยุกต์ใช้งานสามารถทำได้ซับซ้อนมากขึ้นโดยมีตัวอย่างแอพพลิเคชันต่างๆที่เหมาะแก่การพัฒนาโดยบอร์ดคอม86 ดังนี้

Industrial contro l and Automation

ตัวอย่างของอุปกรณ์ควบคุมทางด้านโรงงานเช่นการควบคุมเครื่องจักรจากระยะไกล การตรวจวัดสภาพแวดล้อมต่างๆภายในโรงงานและส่งกลับไปยังศูนย์กลางเป็นต้น

USB peripheral

อุปกรณ์ USB สามารถพัฒนาโดยใช้บอร์ดคอม86 โดยสามารถเชื่อมต่อเข้ากับคอมพิวเตอร์ ด้วยความร็ว 12 Mbps โดยไม่ต้องมีวงจรเพิ่มเติม

Embedded web server

Embedded web server สามารถสร้างได้โดยใช้บอร์ดคอม86โดยไม่ต้องใช้อุปกรณ์อื่นเพิ่มเติม โดยสามารถสื่อสารกับภายนอกผ่านทาง Ethernet port ที่อยู่บนบอร์ดคอม86

Robotics

บอร์ดคอม86 มีไมโครคอนโทรลเลอร์ที่มีประสิทธิภาพสูงในการทำงาน การพัฒนาซอฟต์แวร์สามารถพัฒนาได้ซับซ้อนมากยิ่งขึ้น การเชื่อมต่อขยายทางด้านฮาร์ดแวร์ต่างๆสามารถทำได้โดยง่าย จึงเหมาะสำหรับการพัฒนาหุ่นยนต์ที่มีการทำงานที่ซับซ้อน

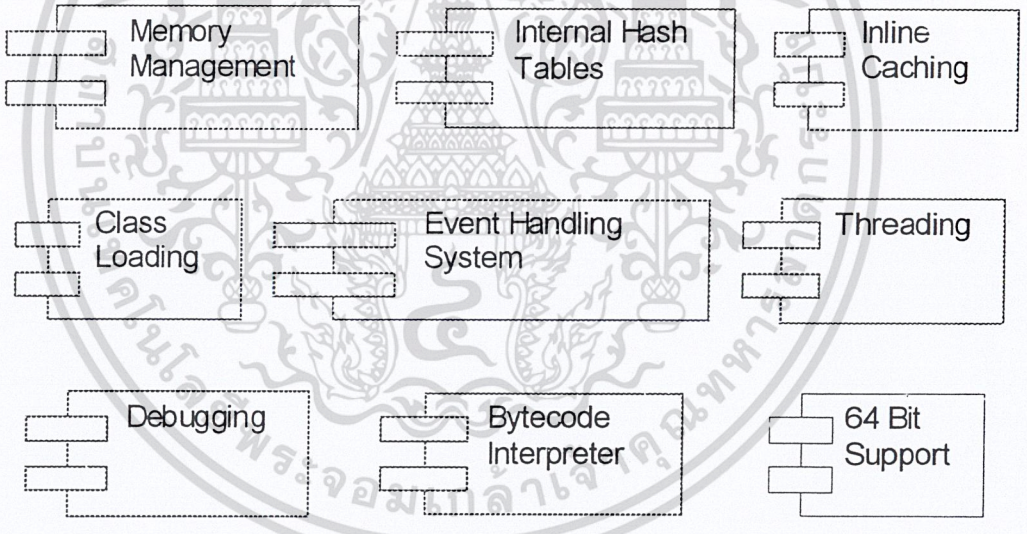
นอกจากนี้ตัวไมโครคอนโทรลเลอร์ยังรองรับการนำไปสร้างเป็นอุปกรณ์ต่างๆที่เกี่ยวข้องกับงานทางด้านการสื่อสารต่างๆ เช่น ISDN Modem and Terminal Adaptor, Low-end Router, Line card application, xDSL application, Digital corded phone เป็นต้น ซึ่งสามารถบอร์ดคอม86ในการออกแบบพัฒนาได้เช่นกัน

สำหรับโครงงานนี้ก็เป็นการพัฒนาให้ภาษาจาวาทำงานได้บนบอร์ดคอม 86 ดังนั้น เมื่อพัฒนาได้แล้ว ก็จะทำให้สามารถพัฒนาจาวาแอพพลิเคชั่นให้ทำงานบนคอม86 ได้อย่างหลากหลาย ซึ่งจะมีผลทำให้เกิดกระแสการพัฒนาจาวาแอพพลิเคชั่น บอร์ดคอม 86 มีความสามารถสูงขึ้นและทำให้บอร์ดคอม86 มีการใช้งานกันแพร่หลายมากยิ่งขึ้นอีกด้วย

บทที่ 3
โครงสร้างและการทำงานของเวอร์ชวลแมชีนขนาดเล็ก (KVM)

ในภาษาจาวานั้น โครงสร้างหลักที่สำคัญที่จำเป็นต้องมีในการใช้งานภาษาจาวานั้นก็คือ จาวาเวอร์ชวลแมชีน ซึ่งส่วนการทำงานหลักของเวอร์ชวลแมชีนขนาดเล็กที่ใช้นั้น จะอยู่ในส่วนที่เป็น VmCommon การทำงานในส่วนนี้ จะเป็นโครงสร้างการทำงานของเวอร์ชวลแมชีนขนาดเล็กที่ไม่ขึ้นอยู่กับระบบปฏิบัติการของอุปกรณ์ที่นำเอาเวอร์ชวลแมชีนขนาดเล็กไปติดตั้ง ดังนั้นในการติดตั้งเวอร์ชวลแมชีนขนาดเล็กลงบนอุปกรณ์ใดๆ ก็ตาม จะต้องนำเอาส่วนนี้ไปใช้แทบทั้งหมด ยกเว้นส่วนที่ระบบปฏิบัติการไม่สามารถทำงานได้จริงๆเช่น ส่วนสนับสนุนการทำงานแบบ 64 บิตไม่จำเป็นที่จะต้องใช้บนระบบปฏิบัติการคอส เป็นต้น ดังที่จะได้กล่าวรายละเอียดต่อไป นอกจากนี้ ในบทนี้ยังได้อธิบายถึงรายละเอียดของส่วนประกอบต่างๆ ของ VmCommon แยกเป็นส่วนการทำงานต่างๆ ดังนี้

3.1 ส่วนประกอบหลักของเวอร์ชวลแมชีนขนาดเล็ก (KVM)



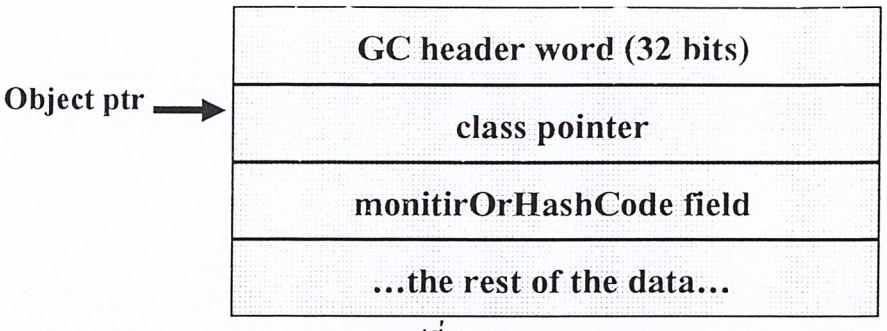
รูปที่ 3-1 Component Diagram

3.2 โครงสร้างรันไทม์ภายในเวอร์ชวลแมชีน (Internal runtime structures)

เป็นโครงสร้างการทำงานภายในของเวอร์ชวลแมชีนขนาดเล็ก (KVM) ที่จัดการเกี่ยวกับรันไทม์ (Runtime) ของจาวาเวอร์ชวลแมชีน ประกอบไปด้วยส่วนหลักๆ 4 ส่วน ดังนี้

3.2.1 โครงสร้างที่เกี่ยวกับคลาส (Class structures)

ทุกๆ ออบเจ็กต์ ในเวอร์ชวลแมชีนขนาดเล็ก (KVM) จะมีโครงสร้างดังรูป



รูปที่ 3-2 Object in KVM

ประกอบไปด้วย ไฟล์ class.c,class.h ประกอบด้วยโครงสร้างต่างๆ ที่ต้องกำหนดไว้ ดังนี้

- CLASS
- INSTANCE_CLASS
- ARRAY_CLASS
- OBJECT
- INSTANCE
- STRING_INSTANCE
- INTENED_STRING_INSTANCE
- ARRAY
- POINTERLIST
- WEAKPOINTERLIST
- BYTEARRAY
- SHORTARRAY

นอกจากนี้ ในส่วนการทำงานเกี่ยวกับคลาสจะต้องทำการกำหนดตัวแปรแบบทั่วไป (Global Variables) โดยต้องทำการกำหนดพอยเตอร์ที่อ้างอิงถึงคลาสต่างๆ ที่สำคัญในภาษาจาวาที่ต้องใช้ในการทำงานของโปรแกรม เช่น Java.Lang.String Java.Lang.Object เป็นต้น

3.2.2 ฟิลด์ (Field)

ฟิลด์เป็นโครงสร้างพื้นฐานที่ถูกใช้ในการแสดงข้อมูลเกี่ยวกับตัวแปร (variables) ค่าคงที่ (constants) และเมธอด (methods) โดยในส่วนรันไทม์นี้ ประกอบด้วยไฟล์ field.c และ field.h ซึ่งจะมีโครงสร้างสำหรับเก็บชนิดต่างๆ ของฟิลด์ และจะมีการสร้างเป็นตารางของฟิลด์เหล่านั้นด้วย โดยตารางของฟิลด์จะถูกสร้างทุกครั้งที่คลาสใหม่ถูกโหลดเข้ามาในเวอร์ชวลแมชีน ตารางของฟิลด์จะมี 2 แบบด้วยกันคือ

- ตารางของฟิลด์(FIELDTABLE)
- ตารางของเมธอด(METHODTABLE)

3.2.3 อินไลน์แคชชิ่ง (Inline Caching)

ในส่วนนี้จะประกอบไปด้วยไฟล์ cache.c และ cache.h เป็นการทำอินไลน์แคชชิ่ง (Inline Caching) ซึ่งจะช่วยให้การทำงานของเวอร์ชวลแมชีนเร็วขึ้น โดยในเวอร์ชวลแมชีนขนาดเล็กนี้จะใช้วิธีการทำอินไลน์แคชชิ่งของ Deutsch-Schiffman Smalltalk-style ที่มีมาตั้งแต่ปี 1980 ในการจัดการการส่งแมสเสจ(messages) และไบต์โค้ด(bytecodes) อื่นๆ ขั้นตอนการทำงานนั้น ถ้าในเวอร์ชวลแมชีนขนาดเล็กถูกกำหนดค่าในไฟล์ machine_md.h ให้อยู่ในโหมดที่ต้องการให้มีการทำอินไลน์แคชชิ่งซึ่งก็คือ ENABLEFASTBYTECODES แล้ว ในเวลารันไทม์ไบต์โค้ดต่างๆก็จะถูกผนึกติดเข้ากับอินไลน์แคชเอ็นทรี (Inline cache entry) เมื่อจาวาแมสเสจส่งไบต์โค้ดออกมา จะทำให้มีพื้นที่หน่วยความจำไม่เพียงพอสำหรับอินไลน์พารามิเตอร์(Inline parameters) ดังนั้น อินไลน์แคชจะทำการจองหน่วยความจำให้และจัดให้มีส่วนที่อ้างอิงถึงอินไลน์แคชเอ็นทรีโดยเฉพาะ ดังรูป

Code before optimization :

BC = "SLOW" bytecode (8 bits)

PARAM = parameter for BC (16 bits)

Code after optimization :

BF = "FAST" bytecode (8 bits)

PARAM = inline cache index (16 bits)

รูปที่ 3-3 Inline Cache in KVM

อินไลน์แคชหลักของเวอร์ชวลแมชีนนั้นจะใช้ชื่อว่า InlineCache ซึ่งอินไลน์แคชจะเป็นอะเรย์ของโครงสร้างแบบ ICACHE ที่มีรูปแบบดังนี้

```
/* ICACHE (allocated in inline cache area) */
struct icacheStruct {
  cell* contents;
  BYTE* codeLoc;
  short origParam;
  BYTE  origInst;
};
```

contents : เป็นข้อมูลที่อยู่ในแคช

codeLoc : code location เป็นพอยเตอร์ที่ชี้ไปที่ตำแหน่งของโค้ดที่อ้างถึงอินไลน์แคชอันนี้

origParam : original parameter เป็นพารามิเตอร์แบบไบต์โค้ดเดิม ซึ่งจะอยู่ในตำแหน่ง codeLoc + 1

origInst : original Instuction เป็นคำสั่งแบบไบต์โค้ดเดิมที่อยู่ในตำแหน่ง codeLoc

โดยในไฟล์ global.h จะมีการประกาศ ICACHE ไว้ คือ

typedef struct icacheStruct* ICACHE;

นอกจากนี้ ในเวอร์ชวลแมชีนขนาดเล็กจะมีการกำหนดจำนวนของอินไลน์แคชไว้ตายตัว โดยแต่ละอินไลน์แคชเอ็นทรีจะมีขนาดอยู่ระหว่าง 12-16 ไบต์ ขึ้นอยู่กับแพลตฟอร์มที่นำเวอร์ชวลแมชีนไปใช้ ดังนั้น เมื่อจำนวนของอินไลน์แคชมีจำกัดเมื่อทำงานไปเรื่อยๆ จึงทำให้พื้นที่ของอินไลน์แคชเต็ม ก็จะต้องมีการนำอินไลน์แคชเดิมกลับมาใช้ใหม่ซ้ำอีกครั้ง โดยจะกลับไปนำอันที่เก่าที่สุดที่อยู่ในส่วนต้นมาใช้ก่อน โดยโค้ดที่อ้างถึงอินไลน์แคชที่ใช้แล้วนั้นจะถูกเก็บแทนที่โค้ดเดิม สรุปก็คือ อินไลน์แคชนั้นจะถูกแก้ไขซ้ำวนไปเรื่อยๆ

เพื่อที่จะหลีกเลี่ยงปัญหาการ์เบจคอลเลคชั่น (garbage collection) ดังนั้น ในเวอร์ชวลแมชีนขนาดเล็กจึงไม่เก็บค่าพอยเตอร์ที่เป็น dynamic heap pointer ไว้ในอินไลน์แคชด้วย

3.2.4 การทำงานเกี่ยวกับเฟรม (Frame)และจัดการเกี่ยวกับเอ็กเซ็พชั่น (Exception Handling)

ส่วนนี้จะเกี่ยวข้องกับไฟล์ frame.c และ frame.h เป็นการทำงานที่จัดการเกี่ยวกับสแต็กเฟรม (Stack Frames) และทำการจัดการเกี่ยวกับเอ็กเซ็พชั่น (Exception Handling) ต่าง ๆ โดยสแต็กเฟรมนั้นจะมีโครงสร้างเรียงจากล่างขึ้นบนดังนี้

1. พารามิเตอร์ที่เมธอดเรียกใช้(Method call parameter) และ ตัวแปรเฉพาะ (Local variables)
2. โครงสร้างแบบเฟรม (frameStruct) มีรายละเอียดดังนี้

```
struct frameStruct {
  FRAME   previousFp;
  BYTE*   previousIp;
  cell*   previousSp;
  METHOD  thisMethod;
  STACK   stack;
  OBJECT  syncObject;
};
```

เฟรมจะถูกจองหน่วยความจำไว้ภายในสแต็กที่ทำงานของเทรดนั้นๆ ตัวแปรต่างๆ มีความหมายคือ

previousFp : เก็บค่าพอยเตอร์ที่ชี้ไปที่เฟรมก่อนหน้านี้ (previous frame pointer)

previousIp : เก็บค่าของโปรแกรมเคาท์เตอร์ก่อนหน้านี้ (previous program counter)

previousSp : เก็บค่าพอยเตอร์ที่ชี้ไปที่สแต็กก่อนหน้านี้ (previous stack pointer)

syncObject : จัดการดูแลออบเจ็ค ถ้าถูกเมธอดเรียกพร้อมกัน

3. โอเปอแรนด์สแต็ก (Operand Stack) เราจะไม่แยกโอเปอแรนด์สแต็กออกมาจากข้อมูล เพราะว่าจำเป็นที่จะต้องให้โอเปอแรนด์ถูกเก็บไว้ที่ชั้นบนสุดของสแต็กเฟรมที่กำลังทำงานอยู่เสมอ

สำหรับการจัดการเกี่ยวกับเอ็กเซ็พชั่น (Exception Handling) นั้น จะมีการทำงานที่ก่อให้เกิดเอ็กเซ็พชั่นหรือข้อผิดพลาด (Errors) จากภายในเวอร์ชวลแมชีนขนาดเล็กเอง อยู่ 3 แบบ คือ

1. raiseException : เป็นข้อผิดพลาดของจาวาที่เกิดขึ้น ตามปกติในเวลาที่ทำงาน โดยจะแสดงเฉพาะข้อผิดพลาดหรือเอ็กเซ็พชั่น ที่สามารถแก้ไขได้

2. fetalError : เป็นข้อผิดพลาดที่เกิดจากจาวา ไม่ได้เกิดขึ้นมาจากเวอร์ชวลแมชีน แต่ก็อาจจะทำให้เกิดความเสียหายต่อเวอร์ชวลแมชีนได้ ต่างจาก raiseException คือเมื่อเกิดขึ้นจะทำให้โปรแกรมหยุดการทำงานเลยทันที

3. fetalVMError : เป็นข้อผิดพลาดที่เกิดจากความผิดปกติภายในเวอร์ชวลแมชีนเองหรือเป็นข้อผิดพลาดที่เกี่ยวข้องกับเวอร์ชวลแมชีน

3.3 การสร้างแฮชเทเบิล (Internal Hash Tables)

มี 3 ตาราง คือ

- Class Table : เก็บข้อมูลของคลาสที่โหลดเข้ามา
- InternStringTable : เก็บค่าสตริงที่ใช้
- UTFStringTable : เก็บค่าสตริงที่ใช้

3.4 การโหลดคลาส (Class Loader)

ประกอบไปด้วยไฟล์ loader.c , loader.h , loaderFile.c ซึ่งมีรายละเอียดต่างๆ ดังนี้

3.4.1 loader.c และ loader.h

เป็นการกำหนดรายละเอียดต่างๆ ภายในเวอร์ชวลแมชีนขนาดเล็กที่จำเป็นต้องใช้ในการโหลดไฟล์จาวาที่เป็นไฟล์ที่มีนามสกุลเป็น .class และสร้างสิ่งต่างๆ ที่จะต้องมีในรันไทม์ของเวอร์ชวลแมชีน โครงสร้างของข้อมูลที่ใช้ในการโหลดคลาสคือ FILEPOINTER เป็นโครงสร้างที่ใช้สำหรับอ้างอิงเพื่อที่จะเปิดไฟล์จากหน่วยความจำ

FILEPOINTER

```
struct filePointerStruct;
typedef struct filePointerStruct *FILEPOINTER;
```

ฟังก์ชั่นการทำงานในการโหลดคลาส ได้แก่

- InitializeClassLoading & FinalizeClassLoading
- openClassfile & closeClassfile
- loadBytesNoEOFCheck : n bytes
- loadByte, loadShort, loadCell : 1,2,4 bytes
- loadBytes : n bytes
- skipBytes, getBytesAvailable : n bytes

ในไฟล์ loader.c จะมีการกำหนดโอเปอเรชั่นต่างๆ ที่ใช้สำหรับอ่านไฟล์จาวาที่เป็น .class ไว้ซึ่งบางส่วนจะเป็นการทำงานที่เกี่ยวข้องกับชนิดของระบบที่ใช้ (platform-dependent) ทำให้ต้องมีการกำหนดค่าตามระบบที่จะนำเคเวอร์ชวลแมชีนมาใช้ซึ่งจะแยกนำไปเก็บไว้ในส่วนของไฟล์ loaderFile.c

3.4.2 loaderFile.c

เป็นโครงสร้างและโอเปอเรชั่นที่ใช้ในการโหลดไฟล์จาวาที่เป็น .class จากระบบไฟล์ที่มีอยู่ซึ่งจะเป็นฟังก์ชั่นในไฟล์นี้จะมีการเรียกใช้ส่วนที่ได้กำหนดให้ขึ้นอยู่กับระบบที่ใช้งาน (platform-dependent) เช่นรูปแบบของการเปิดไฟล์ รูปแบบของการอ่านข้อมูลในไฟล์ ซึ่งในส่วนนี้ จะมีการโหลดไฟล์เข้ามาและเช็คด้วยว่าไฟล์ที่โหลดเข้ามานั้นเป็นแบบไหน เป็นไดเรคทอรี่ ไฟล์ .jar หรือไฟล์ .class โดยเนื้อหาในไฟล์นี้โดยละเอียดจะอยู่ในบทต่อไป

3.5 การจัดการเกี่ยวกับหน่วยความจำ (Memory management)

3.5.1 ฮีป (Heap) และการจองหน่วยความจำ (Memory Allocation)

การทำงานในส่วนนี้จะถูกกำหนดไว้ในไฟล์ collector.c ความหมายของฮีป(Heap) คือ ฮีปเป็นเหมือนหน่วยความจำหลักของอุปกรณ์ที่เวอร์ชวลแมชีนใช้ทำงานอยู่ ในอุปกรณ์ขนาดเล็กที่ใช้เวอร์ชวลแมชีนขนาดเล็กมักจะใช้คำว่าฮีป และจะมีการกำหนดค่าเริ่มต้นให้กับฮีป โดยกำหนดขนาดของฮีปได้ตามต้องการและจะทำการจองพื้นที่หน่วยความจำให้ซึ่งจะทำการจองเมื่อเวอร์ชวลแมชีนเริ่มทำงาน(Start Up) แต่จะยังไม่สร้างเป็นพื้นที่หน่วยความจำถาวร (Permanent Space) เพื่อให้สามารถขยายขนาดของพื้นที่ได้ตามต้องการก่อน โดยที่ขนาดของฮีปนั้นจะเป็นค่าตัวเลขที่หารด้วย 4 ลงตัว และมีค่าอยู่ระหว่าง 16 กิโลไบต์ ถึง 64 เมกะไบต์ ในส่วนการจองหน่วยความจำก็ได้มีการกำหนดฟังก์ชั่นการทำงานไว้และมีการกำหนดฟังก์ชั่นการทำงานอื่นๆ ที่เกี่ยวข้องกับการใช้งานและจัดการเกี่ยวกับหน่วยความจำด้วย

3.5.2 การจัดการเกี่ยวกับปัญหาการ์เบจคอลเลคชั่น (Garbage collection)

การจัดการเกี่ยวกับเรื่องนี้ในเวอร์ชวลแมชีนขนาดเล็กจะถูกกำหนดไว้ในไฟล์ garbage.c และ garbage.h การทำการ์เบจคอลเลคเตอร์ในเวอร์ชวลแมชีนขนาดเล็กนั้น มีการพัฒนามาดังนี้ เริ่มต้นจากสปอตเล็ต (Spotless) ที่เป็นเวอร์ชวลแมชีนขนาดเล็กรุ่นแรก การทำการ์เบจคอลเลคเตอร์จะมีพื้นฐานมาจากแบบ Cheney 's elegant ของ ACM ที่ได้เริ่มนำมาใช้ในปี 1970 โดยจะใช้หลักการทำสำเนา (Copying) และหลักการทำซ้ำเป็นรอบ (Iterative) สำหรับการจัดเก็บออบเจ็ค การทำการ์เบจคอลเลคเตอร์แบบนี้ก่อให้เกิดประโยชน์อย่างมากในตอนนั้น เพราะการทำงานแบบนี้ดีกว่าการใช้หลักการของรีเคอร์ซีพ (Recursive) แต่เนื่องจากเป็นการคัดสำเนาไว้ จึงทำให้ต้องใช้พื้นที่หน่วยความจำมากขึ้นเป็น 2 เท่าของพื้นที่ที่โปรแกรมใช้ตามปกติ ดังนั้น การทำการ์เบจคอลเลคเตอร์แบบนี้จึงเกิดปัญหาเมื่อเราจะนำ เวอร์ชวลแมชีนขนาดเล็กไปติดตั้งลงบนอุปกรณ์ที่มีหน่วยความจำที่จำกัด ซึ่งมักจะเป็นอุปกรณ์ขนาดเล็กเช่น โทรศัพท์มือถือ พีดีเอ เป็นต้น

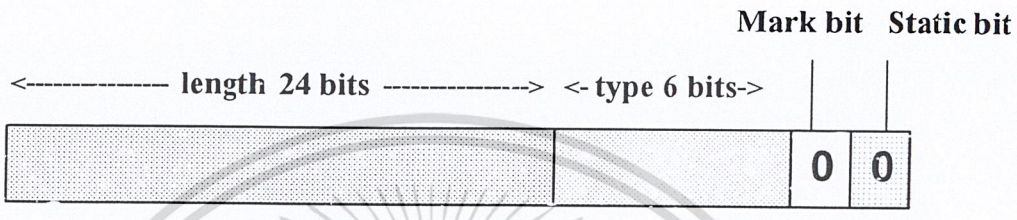
เพื่อที่จะทำให้เวอร์ชวลแมชีนขนาดเล็กสามารถทำงานบนอุปกรณ์ขนาดเล็กได้ ในเวอร์ชวลแมชีนขนาดเล็กเวอร์ชั่น 1.0 จึงเปลี่ยนวิธีทำการ์เบจคอลเลคเตอร์มาใช้วิธีการแบบ straightforward, handle-free , non-moving , mark-and-sweep ดังนั้น ตั้งแต่เวอร์ชวลแมชีนขนาดเล็กเวอร์ชั่น 1.0 จึงไม่สามารถย้ายวัตถุได้ (non-moving) ซึ่งจะทำให้การทำการ์เบจคอลเลคเตอร์ค่อนข้างที่จะทำงานง่ายกว่าในแบบเดิมที่เป็นสปอตเล็ต (Spotless) อย่างไรก็ตาม ตั้งแต่ เวอร์ชวลแมชีนขนาดเล็กเวอร์ชั่น 1.0 จะมีการทำการคืนค่าหน่วยความจำที่ใช้เป็นลิสต์ (list) ที่จองไว้ ซึ่งจะดีกว่าการทำสำเนา (Copying) และการทำการจัดพื้นที่หน่วยความจำให้ใช้พื้นที่น้อยลง (Compact memory) สำหรับเรื่องการจอง

หน่วยความจำให้ออบเจ็คนั้นไม่เร็วกว่าในสปอตเล็ต (Spotless) แต่การทำเมมโมรี่เฟรกเมนเทชั่น (Memory fragmentation) สามารถจะทำให้เวอร์ชวลแมชีนใช้หน่วยความจำได้ทั้งหมดได้ดีกว่าวิธีการทำสำเนาอีกด้วย

การกำหนดโครงสร้างที่เกี่ยวข้องในการทำการ์เบจคอลเลคเตอร์ ได้แก่

3.5.3 ส่วนหัวของออบเจ็ค (Object Header)

โครงสร้างการจองหน่วยความจำของแต่ละออบเจ็คในเวอร์ชวลแมชีนขนาดเล็กจะเป็นดังรูป 4-1 จะเห็นว่ามีส่วนหัวของออบเจ็คที่เก็บค่าความยาวของออบเจ็ค (length) ชนิดของออบเจ็ค (type) และข้อมูลอื่น ส่วนหัวของออบเจ็คนี้จะมีความยาวทั้งหมด 32 บิตหรือ 4 ไบต์นั่นเอง ดังรูป



รูปที่ 3-4 ส่วนหัวของออบเจ็ค (GC Header Word) 32 bits

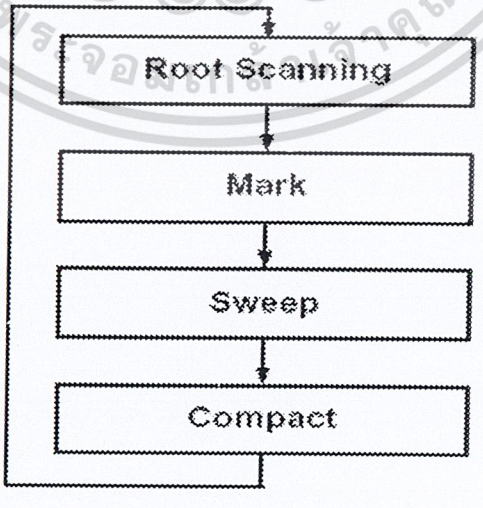
Length : เป็นความยาวของออบเจ็คที่ไม่รวมความยาวของส่วนหัวของออบเจ็ค มีขนาด 24 บิต ถ้า length = 1 แสดงว่าข้อมูลของออบเจ็คมีขนาด 4 ไบต์ ดังนั้น ขนาดของออบเจ็คที่น้อยที่สุดก็คือ ส่วนหัว 4 ไบต์ ส่วนข้อมูล 4 ไบต์ รวมเป็น 8 ไบต์

Type : เป็นชนิดของการ์เบจคอลเลคชั่นของออบเจ็ค มีขนาด 6 บิต

Static bit : เป็นบิตที่ใช้สำหรับบอกว่า ออบเจ็คนี้ได้จองหน่วยความจำไว้ในส่วนที่เป็นสแตติก (Static heab) มีขนาด 1 บิต

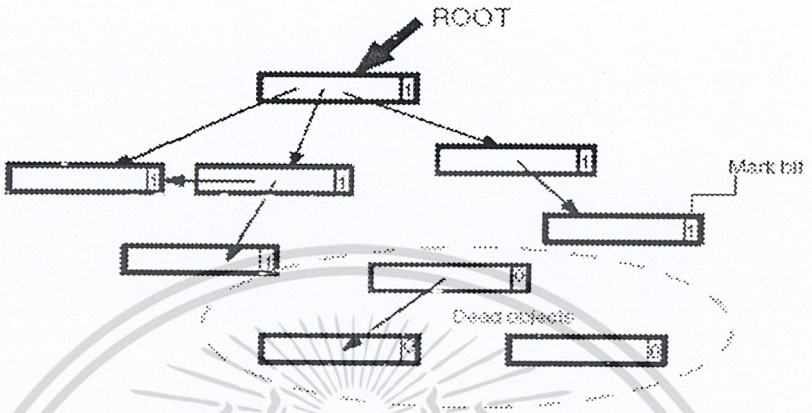
Mark bit : ใช้สำหรับทำเครื่องหมายไว้ว่าออบเจ็คนี้ยังใช้อยู่ (live) ไม่เป็นการ์เบจ (non-garbage) มีขนาด 1 บิต โดยกำหนดค่าไว้ให้เป็น 0 ดังนั้น ทุกออบเจ็คในหน่วยความจำที่ใช้อยู่(heap object) ควรจะมีค่าบิตนี้เป็น 0 เสมอ

3.5.4 ขั้นตอนการทำการ์เบจคอลเลคชั่น



รูปที่ 3-5 ขั้นตอนการทำการ์เบจคอลเลคชั่น

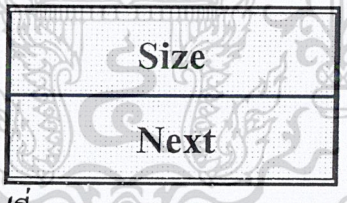
1. Root Scanning : ค้นหารูท
2. Mark : กำหนดค่า marked bit เป็น 1 ถ้ายังใช้ออบเจ็คนั้นอยู่
3. Sweep : ค้นหาออบเจ็คที่เป็นการ์เบจ คือ marked bit เป็น 0 และทำการคืนค่าหน่วยความจำ
4. Compact : รวบรวมพื้นที่ว่างเข้าด้วยกัน



รูปที่ 3-6 Mark and Sweep Algorithm

3.5.5 โครงสร้างในส่วนฟรีลิสต์ (Free List Structures)

ในการทำการ์เบจคอลเลคเตอร์ จะมีการทำฟรีลิสต์ของหน่วยความจำที่ทำการคืนค่าแล้วจากการทำการ์เบจคอลเลคชั่น ดังนั้น ในทุกๆ ส่วนของหน่วยความจำที่ใช้ได้ (Available Memory Chunk) จะมีข้อมูลส่วนหัวของฟรีลิสต์นำหน้าอยู่เสมอ ดังรูป



รูปที่ 3-7 Free List Header in KVM

Size : เป็นขนาดของหน่วยความจำที่มีในส่วนนี้ (amount of memory chunk) มีลักษณะการเก็บค่าเหมือนกับการเก็บข้อมูลส่วนหัวของออบเจ็คที่มี 32 บิต แต่ในส่วนนี้จะใช้มากที่สุดแค่เพียง 24 บิต ส่วน 6 บิตที่เหลือจะต้องกำหนดค่าไว้เป็น 0

Next : เป็นพอยเตอร์ที่ชี้ไปหน่วยความจำส่วนถัดไปในฟรีลิสต์ (pointer to next chunk) ถ้ามีค่าเป็น NIL (0) แสดงว่าเป็นส่วนสุดท้ายของหน่วยความจำ (Last Free Chunk)

ดังนั้น ในแต่ละ Chunk ของหน่วยความจำจะมีความยาวอย่างน้อยที่สุด 2 words ซึ่งก็คือ 64 บิต เพราะว่านี่เป็นขนาดที่น้อยที่สุดที่ต้องใช้เมื่อมีการจองหน่วยความจำให้กับออบเจ็ค

3.5.6 การทำงานในส่วนเท็มโพรารี่รูท (Temporary Root)

เป็นโครงสร้างที่ใช้สำหรับการทำการกำหนด("Pining down") ให้กับออบเจ็คชั่วคราว (Temporary Object) หรือโครงสร้างแบบอื่น ๆ ซึ่งจะถูกจองหน่วยความจำไว้ชั่วคราว ก็คือ จาวาออบเจ็คหรือโครงสร้างในส่วนรันไทม์ของเวอร์ชวลแมชีนยังไม่ถูกเก็บลงหน่วยความจำถาวร

เราจะต้องใช้การทำงานในส่วนเทมโพรารี่รูทเมื่อมีการใช้ส่วนเนทีฟซี (Native C Routine) โดยระบบจะทำการจองหน่วยความจำจากหน่วยความจำหลักของจาวา (Java Heap) ไว้ 2 ออบเจ็ค และจะยังไม่เก็บค่าต่างๆ ของออบเจ็คแรกก่อนที่จะจองหน่วยความจำให้ออบเจ็คที่สอง ถ้าไม่ทำเช่นนั้น ก็จะเกิดปัญหาการ์เบจคอลเลคชั่นขึ้นเมื่อออบเจ็คที่สองจองหน่วยความจำและจะทำให้พอยเตอร์ที่ชี้ไปยังออบเจ็คแรกไม่สามารถใช้งานได้ถูกต้อง ดังตัวอย่าง

TYPICAL PROBLEM SCENARIO :

```
void yourNativeRoutineWrittenInC() {
  // Allocate 100 bytes from Java heap
  char* foo = mallocBytes(100);
  // Allocate another 100 bytes from Java heap
  char* bar = mallocBytes(100);
 /*  ^
      THE SECOND ALLOCATION ABOVE MAY CAUSE A
      GARBAGE COLLECTION TO OCCUR AND THEREFORE
      INVALIDATE 'foo' !!!        */
}
```

จากตัวอย่าง จะเห็นว่าในฟังก์ชั่นที่เกี่ยวกับเนทีฟ (Native Function) ทุกอัน จะต้องมีการเขียนให้ไม่เกิดปัญหาการ์เบจคอลเลคชั่นแบบนี้ โดยที่ควรจะเขียนฟังก์ชั่นในโหมด EXCESSIVE_GARBAGE_COLLECTION แต่การทำงานในโหมดนี้จะทำให้เวอร์ชวลแมชีนขนาดเล็กทำงานช้าลงมากๆ เพราะว่ามันจะต้องทำการแก้ปัญหาการ์เบจคอลเลคชั่น อย่างสมบูรณ์ในทุก ๆ ออบเจ็คที่จองหน่วยความจำไว้

3.6 การทำงานเกี่ยวกับเนทีฟโค้ด (Native Code)

สำหรับการทำงานของจาวา ในแบบอื่นๆ นั้น จะมีการทำงานในส่วนเนทีฟที่เรียกว่า จาวาเนทีฟอินเตอร์เฟส (Java Native Interface = JNI) แต่สำหรับในเวอร์ชวลแมชีนขนาดเล็กนั้นจะไม่สามารถใช้ JNI ได้ แต่จะสนับสนุนการทำงานของ เคเนทีฟอินเตอร์เฟส (K Native Interface = KNI) โดยที่ KNI จะเป็นส่วนหนึ่งของ JNI และรายละเอียดต่างๆ จะถูกกำหนดไว้ใน ไฟล์ KNI.h ซึ่งจะมีรูปแบบการเขียนเนทีฟฟังก์ชั่น ดังนี้

CODE EXAMPLE 2 Handling arguments of native static methods

Java code:

```
static native void
```

```
drawRectangle(int x, int y, int width, int height);
```

Native implementation:

```
static void Java_com_sun_kjava_Graphics_drawRectangle() {
int height = popStack();
int width = popStack();
int y = popStack();
int x = popStack();
windowSystemDrawRectangle(x, y, width, height);
}
```

C type	Macro for popping
char, byte, int, long	popStack()
float	popStackAsType(float)
long64, ulong64	popLong()
double	popDouble()
pointerType	popStackAsType(pointerType)

ตาราง 3-1 Poping arguments from the stack

C type	Macro for pushing
char, byte, int, long	pushStack()
float	pushStackAsType(float)
long64, ulong64	pushLong()
double	pushDouble()
pointerType	pushStackAsType(pointerType)

ตาราง 3-2 Pushing arguments from the stack

3.7 การจัดการเกี่ยวกับอีเวนท์ (Event Handling)

ในการทำงานของจาวาเวอร์ชวลแมชีนนั้น เดิมจะไม่มีการกำหนดรายละเอียดเกี่ยวกับการจัดการเรื่องอีเวนท์ที่เป็นการติดต่อสื่อสารกันระหว่างเวอร์ชวลแมชีนกับระบบปฏิบัติการของอุปกรณ์นั้นๆ แต่ในเวอร์ชวลแมชีนขนาดเล็ก ได้จัด เตรียมส่วนที่จัดการเกี่ยวกับเรื่องนี้ไว้ เป็นเหมือนการรวมการทำงานของเวอร์ชวลแมชีนขนาดเล็กกับระบบจัดการอีเวนท์ของระบบปฏิบัติการบนอุปกรณ์ โดยในเวอร์ชวลแมชีนขนาดเล็กจะมี 4 วิธีด้วยกันในการแจ้งและจัดการเกี่ยวกับอีเวนท์ ดังนี้

1. การแจ้งอีเวนท์แบบซิงโครนัส (Synchronous Notification) หรือ การบล็อกกิ้ง (Blocking)
2. การตรวจหาในจาวาโค้ด (Polling in Java code)
3. การตรวจหาในไบต์โค้ดอินเตอร์พรีเตอร์ (Polling in bytecode interpreter)
4. การแจ้งอีเวนท์แบบอะซิงโครนัส (Asynchronous Notification)

วิธีการจัดการเกี่ยวกับอีเวนท์นี้ จะใช้แตกต่างกันไปตามระบบที่เอาเวอร์ชวลแมชีนขนาดเล็กไปติดตั้ง ขึ้นอยู่กับไลบรารี่ที่เป็นส่วนติดต่อกับผู้ใช้ (User Interface) ว่าสนับสนุนการแสดงผลแบบไหน นอกจากนี้ ก็ยังขึ้นอยู่กับไลบรารี่ที่เป็นการทำงานทางด้านเน็ตเวิร์กอีกด้วย รายละเอียดของวิธีทั้งสี่นั้น มีดังนี้

3.7.1 การแจ้งอีเวนท์แบบซิงโครนัส (Synchronous Notification) หรือ การบล็อกกิ้ง (Blocking)

จะทำการจัดการเกี่ยวกับอีเวนท์โดยเรียกใช้ native I/O หรือฟังก์ชั่นเกี่ยวกับระบบอีเวนท์โดยตรงจากเวอร์ชวลแมชีน ในเวอร์ชวลแมชีนขนาดเล็กนั้นจะมีเพียงแค่ 1 เธรด(Thread)ที่ทำงานในเวอร์ชวลแมชีนเท่านั้น ทำให้จาวาเธรดอื่นๆไม่สามารถทำงานได้ในระหว่างที่เนทีฟฟังก์ชั่นกำลังทำงานอยู่ การทำงานแบบนี้จึงเรียกว่า การบล็อกกิ้ง (Blocking) นั่นเอง วิธีนี้เป็นการทำงานเกี่ยวกับอีเวนท์ที่ง่ายที่สุด แต่ก็ยังไม่ได้รับการยอมรับจริงๆ เนื่องจากในการใช้เนทีฟฟังก์ชั่นนั้น จะมีการออกแบบเป็นลักษณะเฉพาะ จึงต้องดูแลให้เนทีฟฟังก์ชั่นนั้นมีรูปแบบที่สั้นและมีประสิทธิภาพมากที่สุด

3.7.2 การตรวจหาในจาวาโค้ด (Polling in Java code)

บ่อยครั้งที่การจัดการอีเวนท์จะใช้เนทีฟร่วมกันกับจาวาโค้ด เพราะเป็นวิธีที่ง่ายที่สุดที่จะทำให้จาวาเธรดอื่นๆ ทำงานได้ในระหว่างที่กำลังรออีเวนท์นั้นทำงานเสร็จ การใช้วิธีนี้จะต้องใส่จาวาลูป (Loop) ลงในส่วนที่เป็นจาวารันไทม์ไลบรารี่ โดยจะมีโอเปอเรชั่นของ native I/O แบบสั้นๆและจะมีการทำงานวนซ้ำจนกว่าจะทำเสร็จ การตรวจหาในจาวาโค้ดแบบวนซ้ำนี้ (Polling in Java code loop) จะต้องมีการเรียกใช้ Thread.yeild เสมอ

3.7.3 การตรวจหาในไบต์โค้ดอินเตอร์พรีเตอร์ (Polling in bytecode interpreter)

การตรวจหาอีเวนท์ในไบต์โค้ดอินเตอร์พรีเตอร์ (Polling in bytecode interpreter) จะมีการทำการเรียกโอเปอเรชั่นการจัดการอีเวนท์ของเนทีฟมาใช้เป็นระยะๆ การทำงานจะเหมือนกับแบบการแจ้งอีเวนท์แบบซิงโครนัส (Synchronous Notification) แต่จะเป็นส่วนที่เพิ่มเติมมาสำหรับการจัดการอีเวนท์ที่เกี่ยวกับส่วนติดต่อผู้ใช้ที่เป็นแบบกราฟฟิก (Grapchic User Interface) ซึ่งมักจะใช้ในการทำงานบนอุปกรณ์บางอย่างเช่นพีดีเอ

3.7.4 การแจ้งอีเวนท์แบบอะซิงโครนัส (Asynchronous Notification)

การเกิดเหตุการณ์แบบอะซิงโครนัสนั้น มักจะเกิดจากเหตุการณ์ที่มีการทำงาน 2 อย่าง แล้วการทำงานนั้นไม่เป็นไปตามที่ระบบควบคุมได้คิดไว้ ซึ่งเกิดได้หลายรูปแบบ รูปแบบหนึ่งที่มักจะพบก็คือ เมื่อเธรดส่งค่าไปบอกเนทีฟฟังก์ชันว่าจะทำงาน เนทีฟโค้ดก็ได้คาดการว่า เธรดที่ทำงานอยู่นั้น จะทำงานเสร็จสิ้นในอีกไม่นาน และก็จะออกจากลูปการทำงาน ส่งผลให้การทำงานที่รออยู่นั้น สามารถทำงานได้ ตัวแปรคำสั่งจึงเข้าไปแปลคำสั่งของการทำงานต่อไป เวลาต่อมา เธรดการทำงานแรกเกิดมีปัญหา ทำให้การทำงานนั้นยังไม่เสร็จสิ้น ในขณะที่อีกเธรดหนึ่งได้เริ่มทำงานไปแล้ว ในกรณีนี้ ตัวแปรคำสั่งจะทำการแปลคำสั่งเธรดที่ 2 อีกครั้งและเริ่มทำงานใหม่

3.7.5 การทำงานเกี่ยวกับอีเวนท์ในเวอร์ชวลแมชีนขนาดเล็ก

จะมีการทำงานเป็น 2 เลเยอร์ด้วยกัน ในการติดตั้งเวอร์ชวลแมชีนขนาดเล็กลงบนอุปกรณ์ใดๆ จะต้องทำงานทั้งสองส่วน สำหรับในส่วนเลเยอร์แรก มีรายละเอียดดังนี้

ในส่วนบนของอินเตอร์พรีเตอร์ลูปจะมีโค้ดดังนี้

```
If (isTimeToReschedule())
        reschedule();
```

การทำ reschedule นี้ จะมีขั้นตอนการทำงานดังนี้

1. เช็คและค้นหาดูว่ามีจาวาเธรดอื่นๆ ที่ยังทำงานอยู่หรือไม่ และทำการหยุดการทำงานของเวอร์ชวลแมชีนถ้าไม่มีอยู่สักอันเลย
2. เช็คดูว่า ถ้ามีเวลาพอที่จะยอมให้เธรดที่เคยรออยู่ทำงานอีกครั้งได้ไหม ถ้ามีก็ให้เริ่มทำงานโดยอัตโนมัติเลย
3. เช็คดูว่ามีอีเวนท์แบบ I/O เกิดขึ้นหรือไม่
4. พยายามที่จะเปลี่ยนการทำงานไปยังเธรดอื่นๆ

การทำงานในส่วนนี้จะถูกกำหนดไว้ที่ไฟล์ events.h สำหรับเลเยอร์ที่สองจะมีการจัดการเกี่ยวกับอีเวนท์โดยใช้ฟังก์ชั่น GetAndStoreNextKVMEvent(bool_t forever, ulong64 waitUntil)

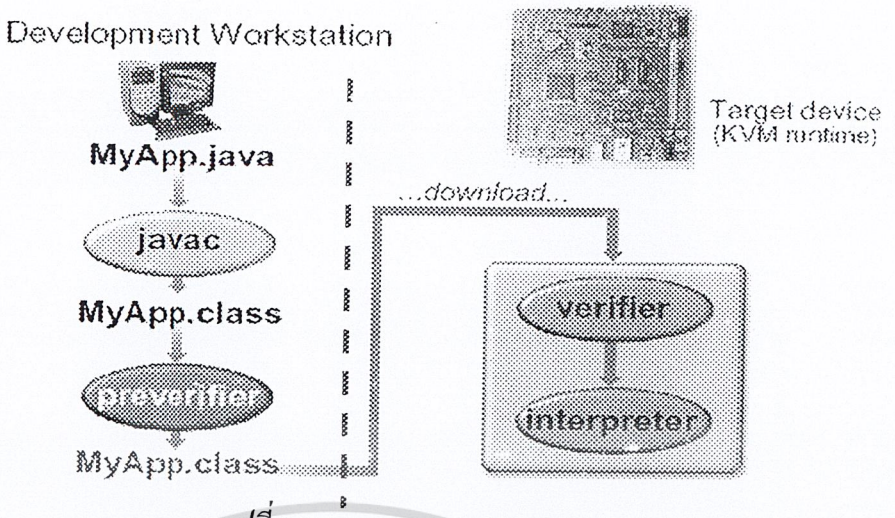
Forever = TRUE : ฟังก์ชั่นนี้จะทำการรออีเวนท์ที่จะเกิดขึ้นนานจนถึงเวลาหนึ่งเท่าที่จำเป็น

Forever = FALSE : ฟังก์ชั่นนี้จะทำการรออีเวนท์ที่จะเกิดขึ้นจนกระทั่งนานเท่าที่จะรอได้

3.8 การตรวจสอบความถูกต้องของไฟล์แบบคลาส (Class File Verification)

การทำงานของเวอร์ชวลแมชีนขนาดเล็ก จะมีส่วนหนึ่งที่ทำหน้าที่เป็นตัวตรวจสอบความถูกต้องของไฟล์คลาส (Class File Verifier) โดยส่วนนี้จะมีอยู่ในจาวาสแตนดาอิดิชั่น (J2SE) ซึ่งจะมีขนาดที่ใหญ่และต้องการพื้นที่สำหรับไลบรารี่โค้ดอย่างต่ำ 50 กิโลไบต์ ต้องการอย่างน้อย 30-100 กิโลไบต์สำหรับไดนามิคแรมใช้ในเวลารันไทม์ นอกจากนี้ ซีพียูที่ใช้ยังต้องสามารถทำงานแบบวิธีอิเทอเรทีฟดาต้าโฟล์ (Iterative Data Flow Algorithm) ได้อีกด้วย

ดังนั้น ในจาวาไมโครอิดิชั่นที่จะนำไปใช้ในอุปกรณ์ขนาดเล็ก จึงต้องดัดแปลงตัวตรวจสอบความถูกต้องของไฟล์คลาส (Class File Verifier) โดยได้ออกแบบใหม่เป็นแบบทูเฟส (Two-phase Class Verifier) ซึ่งจะมีขนาดเล็กกว่าในจาวาสแตนดาอิดิชั่น ในส่วนของรันไทม์นั้นต้องการเพียง 15 กิโลไบต์สำหรับอินเทล (Intel) X86 ไลบรารี่โค้ดและต้องการประมาณร้อยกว่าๆ กิโลไบต์สำหรับไดนามิคแรมในการรันคลาสไฟล์ประเภทต่างๆ ในการทำงาน ตัวตรวจสอบความถูกต้องของไฟล์คลาส (Class File Verifier) นี้จะทำการตรวจสอบไบต์โค้ดแบบอ่านทีละบรรทัดเรียงไปเรื่อยๆ (linear scan) โดยจะไม่มีการใช้วิธีแบบดาต้าโฟล์ (Iterative Data Flow Algorithm) และจะทำงานร่วมกันกับเคเวอร์ชวลแมชีน การทำงานแบบทูเฟสนี้จะเป็นดังรูป



รูปที่ 3-8 Two-phase verification

จากรูป จะเห็นว่าในตัวตรวจสอบความถูกต้องของไฟล์คลาสแบบทูเฟส (Two-Phase Class File Verifier) จะมีการทำงาน 2 ส่วนด้วยกัน คือ

1. ไฟล์คลาสของจาวา จะถูกรันโดยตัวตรวจสอบความถูกต้องเบื้องต้น (Preverifier) ก่อน เพื่อที่จะขยายขนาดคลาสไฟล์โดยการเพิ่มแอททริบิวท์เพื่อที่จะทำให้เวลาที่ใช้ตรวจสอบความถูกต้องตอนรันไทม์เร็วขึ้น การทำงานในส่วนนี้จะทำในขั้นตอนการพัฒนา (Development Workstation) โดยผู้พัฒนาแอพพลิเคชั่นจะทำการเขียนโปรแกรม และคอมไพล์โปรแกรมบนเครื่องคอมพิวเตอร์ทั่วไป การทำการตรวจสอบความถูกต้องเบื้องต้น (Preverifier) ก็จะทำที่ส่วนนี้
2. ในส่วนนี้ จะเป็นการทำงานที่อยู่บนเวอร์ชวลแมชีนขนาดเล็กจริงๆ ส่วนที่ทำการตรวจสอบความถูกต้อง (Verifier) จะทำการตรวจสอบความถูกต้องของคลาสไฟล์ที่เพิ่มแอททริบิวท์จากตัวตรวจสอบความถูกต้องเบื้องต้น (Preverifier) ส่วนนี้จะเป็นขั้นตอนการตรวจสอบจริง ๆ ที่จะทำการตรวจสอบบนอุปกรณ์ที่นำแอพพลิเคชั่นนั้นไปใช้ (Target Device) และจะทำการตรวจสอบในช่วงเวลารันไทม์ของเวอร์ชวลแมชีนที่อยู่บนอุปกรณ์นั้น จะเห็นว่าการตรวจสอบความถูกต้องจะมีประสิทธิภาพมากขึ้น เมื่อทำงานแบบทูเฟสอย่างนี้

ในการตรวจสอบความถูกต้องของคลาสไฟล์นั้น (Class File Verifier) จะมีการเรียกใช้งานในส่วนสแตกแมป และมีการทำงานแบบที่เรียกโปรแกรมย่อยเป็นลำดับ (Inline Subroutines) อีกด้วย

3.9 ส่วนสนับสนุนการทำงานแบบ 64 บิต (64-bits Support)

การทำงานแบบ 64 บิต ก็เป็นสิ่งหนึ่งที่เวอร์ชวลแมชีนขนาดเล็กได้เตรียมไว้ให้สามารถใช้งานได้ แต่สำหรับบางระบบปฏิบัติการหรือบางอุปกรณ์ก็มีข้อจำกัดที่ทำให้ไม่สามารถใช้งานแบบ 64 บิตได้ สำหรับโครงการนี้ เป็นการนำเอาเวอร์ชวลแมชีนขนาดเล็กไปติดตั้งลงบนระบบปฏิบัติดอส ซึ่งโดยปกติแล้วระบบปฏิบัติดอสจะไม่สามารถใช้งานแบบ 64 บิตได้ ดังนั้น ด้วยข้อจำกัดของดอส จึงทำให้โครงการนี้ ไม่จำเป็นต้องใช้ส่วนการทำงานนี้

สำหรับการนำเอาเวอร์ชวลแมชีนขนาดเล็กไปติดตั้งบนระบบปฏิบัติการอื่นๆ นั้น อาจจะต้องใช้ความสามารถของเวอร์ชวลแมชีนขนาดเล็กในส่วนนี้ จึงได้อธิบายรายละเอียดบางส่วน ดังนี้

การทำงานแบบ 64 บิตนี้จะเป็นการทำงานแบบขึ้นกับระบบปฏิบัติการ (Platform-dependent) สามารถกำหนดให้สนับสนุน 64 บิตหรือไม่ก็ได้ โดย

1. ถ้าคอมไพเลอร์สนับสนุนตัวเลขแบบ 64 บิต จะหมายความดังนี้

Type	Description
long64	A signed 64-bit integer.
ulong64	An unsigned 64-bit integer.

ตาราง 3-3 64-bit types

- ต้องกำหนดให้ค่าคงที่ของคอมไพเลอร์ ที่เป็น BIG_ENDIAN หรือ LITTLE_ENDIAN มีค่าไม่เป็น 0 คือ อันใดอันหนึ่งหรือทั้งสองอันมีค่าเป็น 1
- ถ้าใช้คอมไพเลอร์แบบ Gnu C หรือ Solaris C จะต้องกำหนดค่าดังนี้

```
typedef long long long64;
typedef unsigned long long ulong64;
```

- ถ้าใช้คอมไพเลอร์แบบ Visual C/C++ จะต้องกำหนดค่าดังนี้

```
typedef __int64 long64;
typedef unsigned __int64 ulong64;
```

2. ถ้าจะให้คอมไพเลอร์ไม่สนับสนุนตัวเลขแบบ 64 บิต จะต้องกำหนดดังนี้
- กำหนดให้ COMPILER_SUPPORT_LONG มีค่าเป็น 1
- ต้องกำหนดให้ค่าคงที่ของคอมไพเลอร์ ที่เป็น BIG_ENDIAN หรือ LITTLE_ENDIAN อันใดอันหนึ่งเท่านั้น มีค่าเป็น 1

3.10 การทำงานแบบคอนสแตนท์พูล (Constant Pool)

เป็นโครงสร้างที่เก็บข้อมูลของเครื่องหมายต่างๆ ของจาวาคลาสที่ถูกอ้างอิงถึงได้ เป็นเหมือนตัวแสดงรูปแบบของคลาส เมื่อจาวาไบต์โค้ดเข้าถึงข้อมูลและเมธอดโดยตัวชี้ของอินสแตนท์พูลแล้ว รันไทม์คอนสแตนท์พูลก็จะต้องคอยดูแลทุกๆ คลาสในระบบ ขนาดของตัวคอนสแตนท์พูลแต่ละอันต้องมีขนาดเท่ากัน และจะใช้วิธีอินเด็ก(index) หาตำแหน่งของคอนสแตนท์พูลมากกว่าใช้วิธีการค้นหาแบบตามเวลา (time-cosuming lookups) โครงสร้างนั้นเป็น ดังรูป

5	entry[0]
	entry[1]
	entry[2]
	entry[…]
	entry[n-1]

รูปที่ 3-9 Constant Pool Structures

entry[0] จะเก็บค่าของจำนวนของ entries ทั้งหมด +1 ในที่นี้ คือ 4+1 ดังนั้น มี 4 entries

entry[1] จะเก็บค่าของคอนสแตนท์พูลจริงๆ เป็น entry ที่ 1

entry[2] จะเก็บค่าของคอนสแตนท์พูลจริงๆ เป็น entry ที่ 2

entry[n-1] จะเก็บค่าของคอนสแตนท์พูลจริงๆ เป็น entry สุดท้าย ในที่นี้ n = 5

3.11 การเริ่มต้นการทำงานของเวอร์ชวลแมชีน

เป็นการทำงานของไฟล์ StartJVM.c เป็นการกำหนดค่าเริ่มต้นให้กับระบบและเริ่มต้นการทำงาน มีฟังก์ชั่นการทำงาน ดังนี้

- readCommandLineArguments(int argc, char* argv[])

จะอ่านและเก็บค่าอาร์กิวเมนต์ที่คอมมานไลน์ไว้เป็นอะเรย์ของสตริง

- loadMainClass(char* className)

ทำการโหลดเมนคลาส ซึ่งต่างจากคลาสอื่น

- int KVM_Start(int argc, char* argv[])

กำหนดค่าทุกๆ อย่าง ตอนเริ่มต้นทำงาน

- void KVM_Cleanup()

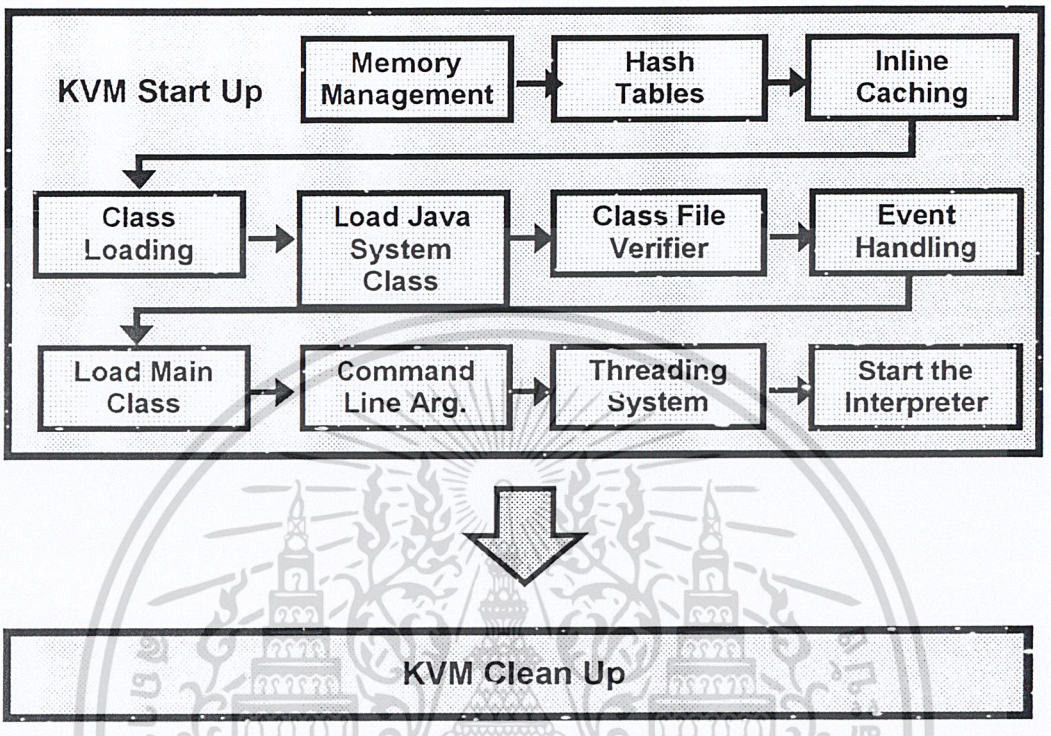
ทำการลบทุกอย่างที่สร้างขึ้นในตอนแรกให้เหมือนเดิม ใช้เมื่อจะเลิกทำงาน

- int StartJVM(int argc, char* argv[])

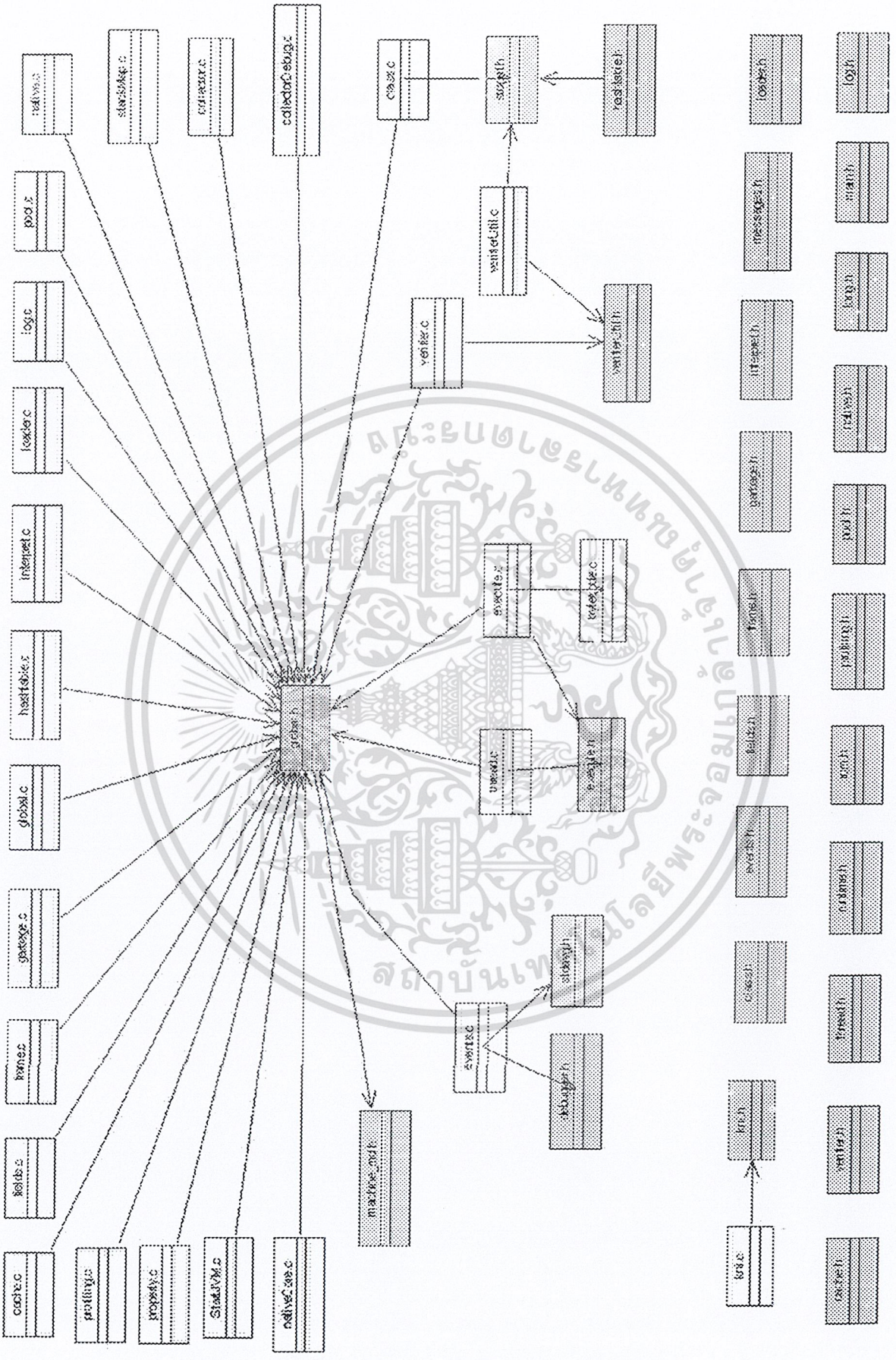
เริ่มต้นการทำงานของเวอร์ชวลแมชีน เริ่มทำงานฟังก์ชั่นเมน

3.12 สรุปขั้นตอนการทำงานในเวอร์ชวลแมชีนขนาดเล็ก (KVM)

เมื่อเริ่มการทำงาน เวอร์ชวลแมชีนขนาดเล็ก(KVM) จะทำงานเป็นขั้นตอนดังรูป และจะทำงานไปเรื่อยจนถึงขั้นตอน KVM Cleanup ก็จะจบการทำงาน



รูป 3-10 ขั้นตอนการทำงานของเวอร์ชวลแมชีนขนาดเล็ก (KVM)



รูป 3-11 โครงสร้างของ VmCommon

บทที่ 4
โครงสร้างและการทำงานในส่วนของ VmExtra

ในโครงสร้างของเวอร์ชวลแมชีนขนาดเล็กนั้น นอกจากโครงสร้างหลักที่เป็นวีเอ็มคอมมอนแล้ว ยังมีส่วนพิเศษของเวอร์ชวลแมชีนขนาดเล็ก นั่นคือ KVM Extra ซึ่งเป็นส่วนที่เพิ่มเติมเข้ามา ให้เวอร์ชวลแมชีนขนาดเล็กมีความสามารถที่มากขึ้น ในส่วนนี้เราจะสามารถนำมาใช้หรือไม่ก็ได้ จะไม่มีผลกับการทำงานหลักของเวอร์ชวลแมชีนขนาดเล็ก

4.1 การทำงานผ่านคอมมานไลน์ (Commandline Environment)

ประกอบไปด้วยไฟล์ main.c ซึ่งมีรายละเอียดต่างๆ ดังนี้

4.1.1 main.c

โปรแกรมโดยทั่วไปนั้น หากจะเริ่มทำงานจะต้องเรียกผ่านฟังก์ชัน main() เป็นตัวแรกก่อน แต่ในเวอร์ชวลแมชีนขนาดเล็กนั้น ไม่จำเป็นที่จะต้องการงานฟังก์ชัน main() ก็ได้ ทั้งนี้เนื่องจากเวอร์ชวลแมชีนขนาดเล็กได้ถูกออกแบบมาให้สามารถทำงานบนอุปกรณ์ที่มีขนาดเล็ก ซึ่งอุปกรณ์บางชนิดเองก็ไม่มีการทำงานผ่านคอมมานไลน์ เพราะฉะนั้นการทำงานผ่านคอมมานไลน์จึงมาอยู่ในส่วนของ VmExtra ซึ่งเราสามารถจะตัดออกไปก็ได้

การทำงานผ่านคอมมานไลน์นั้น จะจัดการเพียงเรื่องการแสดงผลออก การรับค่าคำสั่งผ่านทางคอมมานไลน์เท่านั้น ส่วนการเริ่มต้น หรือหยุดการทำงานของเวอร์ชวลแมชีนนั้น จะถูกกำหนดไว้ที่ไฟล์ VmCommon/src/StartJVM.c แล้ว การทำงานผ่านคอมมานไลน์มีตัวแปรที่สำคัญๆดังนี้

- JamEnabled กำหนดให้เวอร์ชวลแมชีนขนาดเล็กทำงานแจมด้วยหรือไม่ หากไม่ให้ตั้งเป็น FALSE แต่หากต้องการเรียกใช้แจมควรตั้งเป็น TRUE (ค่าปกติคือ FALSE)
- JamRepeat กำหนดให้เวอร์ชวลแมชีนขนาดเล็กทำงานแจมด้วยและมีการทำงานแจมย้อนกลับมาได้อีกครั้ง หากไม่ต้องการให้ตั้งเป็น FALSE แต่หากต้องการให้ตั้งเป็น TRUE (ค่าปกติคือ FALSE)
- RequestedHeapSize กำหนดขนาดของฮีปที่จะเรียกใช้ (ค่าปกติคือ DEFAULTHEAPSIZE)
- UserClassPath กำหนดค่าของคลาสพาธที่เราต้องการใช้เรียกคลาสไฟล์

การทำงานผ่านคอมมานไลน์มีฟังก์ชันที่สำคัญดังนี้

- printHelpText(); ทำการแสดงข้อความช่วยเหลือต่างๆออกมาทางคอมมานไลน์

4.2 การโหลดคลาส(Class Loader)

ประกอบไปด้วยไฟล์ loadfile.c ซึ่งมีรายละเอียดต่างๆ ดังนี้

4.2.1 loadfile.c

เนื่องจากบนอุปกรณ์ขนาดเล็กหลายตัว มีขนาดของหน่วยความจำและเนื้อที่ภายในอุปกรณ์ที่น้อยมาก ซึ่งทำให้บางอุปกรณ์จะไม่มีการทำงานเรื่องระบบไฟล์ หรืออาจจะมีไฟล์โปรแกรมที่ใช้เพียงโปรแกรมเดียวเท่านั้น ด้วยเหตุนี้เวอร์ชวลแมชีนขนาดเล็กจึงทำการแยกการทำงานของคลาสไฟล์กับการโหลดคลาสไฟล์มาใช้ออกจากกัน

แต่ด้วยความที่อุปกรณ์ส่วนใหญ่มักจะมีการทำงานระบบไฟล์อยู่แล้ว การใช้ loadfile.c จึงไม่นิยมตัดออกจากการสร้างเวอร์ชวลแมชีนขนาดเล็ก และมีข้อสังเกตุว่า ทางบริษัทซันได้เขียนโปรแกรมให้หลายฟังก์ชันใน loader.c จะเรียกใช้งานจาก loadfile.c ซึ่งมีผลให้หากเราจะต้องตัด loadfile.c ออกจากเวอร์ชวลแมชีนขนาดเล็กจริง จะต้องเขียนฟังก์ชันอีกหลายอย่างขึ้นมาเองด้วย

การโหลดไฟล์มาใช้งานนั้น จะมีการใช้ตัวอักษรที่น่าสนใจ 2 ตัว ได้แก่

- 'd' หมายถึงไดเรกทอรี
- 'j' หมายถึง JAR ไฟล์ หรือไฟล์ที่บีบอัดข้อมูลไว้(zip file) ซึ่งไฟล์นี้จะประกอบไปด้วยข้อมูลแล้วตามด้วย 3 ช่องว่างที่เราไม่จำเป็นจำต้องสนใจ แล้วตามด้วยข้อมูลตำแหน่งเริ่มต้นของโลคอลเฮดเดอร์ (first Local Header) ขนาด 4 ไบต์ แล้วตามด้วยข้อมูลตำแหน่งเริ่มต้นของเซ็นทรัลเฮดเอดร์ (first Central Header) แล้วตามด้วยชื่อของไฟล์ที่บีบอัด

การโหลดคลาสมีตัวแปรที่สำคัญๆดังนี้

- ClassPathTable เป็นตารางที่เก็บค่าตัวชี้วัตถุ (object pointer)
- MaxClassPathTableLength แสดงค่าความยาวสูงสุดของคลาสพาธที่มีในตาราง

การโหลดคลาสมีฟังก์ชันที่สำคัญๆดังนี้

- InitializeClassLoading เป็นการกำหนดค่าเริ่มต้นต่างๆเพื่อใช้ในการโหลดคลาสไฟล์
- OpenClassFile ทำการอ่านค่าข้อมูลในไฟล์แล้วส่ง FILEPOINTER กลับไป
- OpenResourceFile ทำการอ่านค่าข้อมูลในทรัพยากรที่มี(resource) แล้วส่ง FILEPOINTER กลับไป
- openClassfileInternal เป็นฟังก์ชันที่เรียกใช้มาจากฟังก์ชัน openClassfile() ทำการตรวจสอบดูว่าไฟล์นั้นเป็นไฟล์ที่สามารถอ่านค่ามาได้หรือไม่ และส่ง FILE* กลับไป
- loadByte(), loadShort(), loadCell(), loadBytes, skipBytes() อ่านข้อมูลต่อไป 1,2,4 หรือ n ไบต์
- closeClassfile ทำการปิดคลาสไฟล์ ทำการตรวจสอบให้แน่ใจว่าเราได้อ่านข้อมูลมาจนถึงจุดสุดท้ายของคลาสไฟล์แล้ว
- FinalizeClassLoading ทำการปิดการทำงานทั้งหมดของการโหลดคลาสไฟล์

4.3 การจัดการเกี่ยวกับหน่วยความจำ (Memory Management)

ประกอบไปด้วยไฟล์ fakeStaticMemory.c ซึ่งมีรายละเอียดต่างๆ ดังนี้

4.3.1 fakeStaticMemory.c

การจัดการหน่วยความจำใน fakeStaticMemory.c นั้น เรียกอีกอย่างหนึ่งว่าวิธี "simonizing" ซึ่งออกแบบมาเพื่อใช้สำหรับระบบปฏิบัติการปาล์ม(Palm) หรือสำหรับอุปกรณ์ที่มีขนาดของหน่วยความจำจำกัด แต่มีเนื้อที่เก็บข้อมูลมาก

การทำ simonizing คือการนำข้อมูลที่ปกติจะเก็บเอาไว้ในหน่วยความจำผันแปร (Dynamic) มาเก็บไว้ที่เก็บข้อมูลคงที่(static/storage memory) ซึ่งวิธีนี้จะช่วยให้เวอร์ชวลแมชีนขนาดเล็กสามารถใช้งานฮีปได้มากขึ้น การจะเปิดการจัดการหน่วยความจำแบบ fakeStaticMemory นั้น จะต้องมีการตั้งค่า "USESTATIC" เป็น 1 หรือ TRUE ก่อน

การใช้หน่วยความจำรูปแบบนี้ ไม่มีความจำเป็นเลยสำหรับระบบปฏิบัติการวินโดว์และยูนิกซ์ เพราะมีการจัดการเรื่องหน่วยความจำได้ดีอยู่แล้ว และโดยทั่วไปอุปกรณ์ที่ใช้ระบบปฏิบัติการนี้ก็มักจะมีหน่วยความจำที่มากอยู่แล้ว

การจัดการกับหน่วยความจำมีตัวแปรที่สำคัญๆดังนี้

- memoryStart จุดเริ่มต้นของหน่วยความจำ ค่านี้จะเปลี่ยนไปเมื่อค่าหน่วยความจำที่เราต้องการใช้มีค่าเปลี่ยนไป
- memoryOffset
- pageSize เก็บขนาดของหน้า หรือขนาดของหน่วยความจำที่จะทำการเก็บข้อมูลไว้

การจัดการกับหน่วยความจำมีฟังก์ชันที่สำคัญๆดังนี้

- modifyStaticMemory เปลี่ยนแปลงแก้ไขข้อมูลหน่วยความจำถาวร
- mallocStaticBytes ทำการร้องขอพื้นที่จากหน่วยความจำถาวร
- FinalizeStaticMemory ยกเลิกการร้องขอพื้นที่จากหน่วยความจำถาวรทั้งหมด

4.4 ฟังก์ชันสนับสนุนการทำอะซิงโครนัส (Function Support Asynchronous)

ประกอบไปด้วยไฟล์ async.c และ async.h ซึ่งมีรายละเอียดต่างๆ ดังนี้

4.4.1 async.c และ async.h

การจัดการกับเหตุการณ์ต่างๆในเวอร์ชวลแมชีนขนาดเล็กนั้น นอกจากการทำงานปกติ(Synchronouse notification) การตั้งวาวโค้ด(Polling) การดึงข้อมูลในระดับไบต์โค้ด(Polling in the bytecode interpreter) แล้ว ยังมีการควบคุมเหตุการณ์อีกอย่างหนึ่งคือ การทำอะซิงโครนัส

การเกิดเหตุการณ์แบบอะซิงโครนัสนั้น มักจะเกิดจากเหตุการณ์ที่มีการทำงาน 2 อย่าง แล้วการทำงานนั้นไม่เป็นไปตามที่ระบบควบคุมได้คิดไว้ ซึ่งเกิดได้หลายรูปแบบ รูปแบบหนึ่งที่มักจะพบก็คือ เมื่อเธรดส่งค่าไปบอกเนทีฟฟังก์ชันว่าจะทำงาน เนทีฟโค้ดก็ได้คาดการว่า เธรดที่ทำงานอยู่นั้น จะทำงานเสร็จสิ้นในอีกไม่นาน และก็จะออกจากลูปการทำงาน ส่งผลให้การทำงานที่รออยู่นั้น สามารถทำงานได้ ตัวแปรคำสั่งจึงเข้าไปแปลคำสั่งของการทำงานต่อไป เวลาต่อมา เธรดการทำงานแรกเกิดมีปัญหา ทำให้การทำงานนั้นยังไม่เสร็จสิ้น ในขณะที่อีกเธรดหนึ่งได้เริ่มทำงานไปแล้ว ในกรณีนี้ ตัวแปรคำสั่งจะทำการแปลคำสั่งเธรดที่ 2 อีกครั้งและเริ่มทำงานใหม่

การเกิดเหตุการณ์อะซิงโครนัสนั้น มักจะเกิดจากการผิดพลาดในการเขียนโปรแกรม ซึ่งไม่เกิดขึ้นบ่อยนัก หากเราต้องการให้เวอร์ชวลแมชีนขนาดเล็กที่เราใช้ มีขนาดที่เล็ก เราก็สามารถเลิกการทำงานนี้ออกได้

การทำงานอะซิงโครนัสมีฟังก์ชันที่สำคัญๆดังนี้

- AcquireAsyncIOCB ทำการตัวควบคุมการทำงานอะซิงโครนัส I/O
- ReleaseAsyncIOCB , FreeAsyncIOCB , AbortAsyncIOCB ทำการปล่อยหรือยกเลิกตัวควบคุมการทำงานอะซิงโครนัส I/O
- ActiveAsyncOperations ส่งค่าของจำนวนการทำงานที่ทำอยู่ในตอนนี้
- InitalizeAsynchronousIO เริ่มต้นการทำงาน
- RequestGarbageCollection พยายามค้นหาโค้ดที่ไม่ปลอดภัย

4.5 การทำดีบั๊กในโปรแกรมจาวา (Java Level Debugger)

ประกอบไปด้วยไฟล์ debugger.c , deebugger.h , debuggerInputStream.c , debuggerOutputStream.c , debuggerSocketIO.c , debuggerCommands.h , debuggerStreams.h ซึ่งมีรายละเอียดต่างๆ ดังนี้

4.5.1 debugger.c , debugger.h

การทำดีบั๊กนั้น มีส่วนช่วยให้เราเข้าใจและเข้าไปดูการทำงานของโปรแกรมที่ทำงานอยู่ได้ดีขึ้นมาก ในเวอร์ชวลแมชีนขนาดเล็กนั้น เมื่อเราเปิดการใช้งานดีบั๊กแล้ว เวอร์ชวลแมชีนขนาดเล็กจะทำการแทรกส่วนของการแสดงข้อความเข้ามาเมื่อเราแสดงผลการทำงาน ซึ่งในส่วนของการใช้งานดีบั๊กในเวอร์ชวลแมชีนขนาดเล็กนั้น จะมีอธิบายในบทถัดไป

การจะให้เวอร์ชวลแมชีนขนาดเล็กใช้งานดีบั๊กได้นั้น จะต้องทำการตั้งค่า "ENABLE_JAVA_DEBUGGER." เป็น 1 หรือ TRUE

debugger.c และ debugger.h มีฟังก์ชันที่สำคัญๆดังนี้

- setEvent_VMInit จะถูกเรียกเมื่อเวอร์ชวลแมชีนทำงาน เพื่อตั้งค่าต่างๆ
- setEvent_VMDeath จะถูกเรียกเมื่อเวอร์ชวลแมชีนมีข้อผิดพลาดในการทำงาน และจะทำการจัดการค่าตัวแปรต่างๆ
- isLegalOffset ตรวจสอบดูว่าค่าออฟเซตที่ใช้นั้น ถูกต้องตรงกับโค้ดที่ทำงานหรือไม่
- isValidThread , isValidJavaThread ตรวจสอบดูว่าเธรดที่ทำงานอยู่นั้น ยังทำงานอยู่หรือไม่
- addToDebugRoot เพิ่มออบเจ็กต์นี้เข้าไปในการทำงานดีบั๊ก
- getObjectID สร้างหรือค้นหาไอดี(ID) ให้กับออบเจ็กต์นี้
- getObjectPtr นำค่าไอดีที่ได้มาค้นหาออบเจ็กต์ที่ถูกต้องกับไอดีนี้
- installBreakpoint ทำการเพิ่มจุดหยุดของโค้ด(Breakepoints)ลงไป
- findBreakEventAndOpcode ค้นหาจุดหยุดของโค้ด
- setNotification กำหนดให้ทำการดีบั๊กเฉพาะวิธีใดวิธีหนึ่ง (วิธีการดีบั๊กแบบต่างๆจะอยู่ในบทถัดไป)
- getMethodIndex ค้นหาค่าดัชนี(index) ของเมธอดต่างๆจากตารางเมธอด
- handleSingleStep ตัดสินใจว่า เมื่อใดที่ควรจะทำการสเต็ปปิ้ง(Stepping)
- VirtualMachine_AllClasses ส่งค่าไอดีของคลาสทั้งหมดไปที่ดีบั๊ก
- VirtualMachine_AllThreads ส่งค่าไอดีของเธรดทั้งหมดไปที่ดีบั๊ก
- createEvent สร้างค่าอีเวนต์ขึ้นมา
- clearEvent ยกเลิกค่าอีเวนต์ทั้งหมดหรือตัวใดตัวหนึ่ง

4.5.2 debuggerInputStream.c , debuggerOutputStream.c , debuggerStreams.h , debuggerSocketIO.c

ทำงานเรื่องการดีบั๊กที่เกี่ยวกับการทำงานผ่านเน็ตเวิร์ค ซึ่งจะมีประโยชน์อย่างมากเมื่อโปรแกรมภาษาจาวาของเรามีการเชื่อมต่อกับเน็ตเวิร์ค การทำดีบั๊กจะช่วยให้เราทราบสถานะของการรับส่งข้อมูลอยู่ตลอดเวลา

การทำดีบั๊กเกี่ยวกับเน็ตเวิร์คมีรายละเอียดต่างๆดังนี้

- debuggerInputStream.c จะทำการดีบั๊กในส่วนของการรับข้อมูลเข้ามาผ่านเน็ตเวิร์ค
- debuggerOutputStream.c จะทำการดีบั๊กในส่วนของการส่งข้อมูลออกไปผ่านเน็ตเวิร์ค
- debuggerSocketIO.c ทำงานอยู่ภายใต้เน็ตเวิร์คซ็อกเก็ตสำหรับจาวา
- debuggerStreams.h จัดการตั้งค่าตัวแปรที่จำเป็นต่างๆ

4.5.3 debuggerCommands.h

ทำการสร้างตารางจัดเรียงคำสั่งต่างๆ เพื่อส่งไปให้ดีบั๊กทราบว่า เมื่อเราต้องทำการดีบั๊กคำสั่งนี้จะต้องเรียกฟังก์ชันใดก่อน ฟังก์ชันใดหลัง

4.6 การทำงานเกี่ยวกับเครือข่าย (Networking)

ประกอบไปด้วยไฟล์ commProtocol.c , commProtocol.h , datagramProtocol.c , datagramProtocol.h , socketProtocol.c , socketProtocol.h , resource.c ,ซึ่งมีรายละเอียดต่างๆ ดังนี้

4.6.1 commProtocol.c , commProtocol.h

เป็นฟังก์ชันการทำงาน เพื่อให้เวอร์ชวลแมชีนขนาดเล็กสามารถทำงานติดต่อกับเน็ตเวิร์คได้ โดยผ่านทางซีเรียลพอร์ต ซึ่งการใช้งานนี้จะต้องใช้ร่วมกับไฟล์ใน VmPort เพื่อให้สามารถสื่อสารผ่านทางระบบปฏิบัติการได้

การทำงานผ่านเน็ตเวิร์คส่วนใหญ่จะทำงาน โดยเรียกใช้คลาสที่มีอยู่แล้ว ซึ่งเราสามารถดูการทำงานได้จาก API ที่มีในหนังสือทั่วไป โดยคลาสที่เรียกมาจะมาจาก Java_com_sun_cldc_io_j2me_comm_Protocol_native

commProtocol.c และ commProtocol.h มีฟังก์ชันที่สำคัญๆดังนี้

- Java_com_sun_cldc_io_j2me_comm_Protocol_native_1openByNumber ทำการเชื่อมต่อโดยใช้หมายเลขระบุการเชื่อมต่อ อาจจะเป็น IP Address ก็ได้
- Java_com_sun_cldc_io_j2me_comm_Protocol_native_1openByName ทำการเชื่อมต่อโดยใช้ชื่อระบุการเชื่อมต่อ อาจจะเป็นชื่อโฮสต์หรือชื่อของเครื่องก็ได้
- Java_com_sun_cldc_io_j2me_comm_Protocol_native_1close ปิดการเชื่อมต่อ
- Java_com_sun_cldc_io_j2me_comm_Protocol_native_1readBytes อ่านข้อมูลที่ส่งมาเป็นไบต์
- Java_com_sun_cldc_io_j2me_comm_Protocol_native_1available ตรวจสอบการทำงานว่ามีการส่งข้อมูลอยู่หรือไม่
- Java_com_sun_cldc_io_j2me_comm_Protocol_native_1writeBytes เขียนข้อมูลหรือส่งข้อมูลไปเป็นไบต์
- raiseExceptionWithString แสดงข้อผิดพลาดออกมาทางข้อความ

4.6.2 datagramProtocol.c , dataramProtocol.h

เป็นฟังก์ชันการทำงาน เพื่อให้เวอร์ชวลแมชีนขนาดเล็กสามารถทำงานติดต่อกับเน็ตเวิร์คโดยใช้ดาต้าแกรม (Datagram) ได้

datagramProtocol.c และ datagramProtocol.h มีฟังก์ชันที่สำคัญๆดังนี้

- setSocketHandle จัดการตั้งค่าต่างๆ ที่จะใช้ในการทำดาต้าแกรม
- getSocketHandle ร้องขอค่าติดตั้งต่างๆของดาต้าแกรม

- receive0 รับดาต้าแกรม
- close ปิดการทำงานดาต้าแกรม
- getMaximumLength รับค่าขนาดสูงสุดของดาต้าแกรม
- getNominalLength รับค่าขนาดของดาต้าแกรมตามปกติ
- getHostByAddr แปลงค่า IP Address มาเป็นชื่อของโฮสต์
- getIpNumber แปลงชื่อของโฮสต์มาเป็น IP Address

4.6.3 socketProtocol.c , socketProtocol.h

เป็นฟังก์ชันการทำงาน เพื่อให้เวอร์ชวลแมชีนขนาดเล็กสามารถทำงานติดต่อกับเน็ตเวิร์คโดยใช้ซ็อกเก็ต(socket) ได้

socketProtocol.c และ socketProtocol.h มีฟังก์ชันที่สำคัญๆดังนี้

มีฟังก์ชันที่สำคัญๆดังนี้

- setNonBlocking แปลงชื่อของโฮสต์มาเป็น IP Address
- getIpNumber แปลงค่า IP Address มาเป็นชื่อของโฮสต์
- open0 เปิดทีซีพีซ็อกเก็ต(TCP Socket)
- read0 อ่านข้อมูลจากทีซีพีซ็อกเก็ต
- write0 ส่งข้อมูลผ่านทีซีพีซ็อกเก็ต
- available0 ส่งค่าจำนวนไบต์ที่เหลืออยู่
- close0 ปิดการทำงานทีซีพีซ็อกเก็ต

ฟังก์ชันการทำงานของเซิร์ฟเวอร์

- open เปิดการทำงานผ่านทีซีพีซ็อกเก็ต
- accept รับและเริ่มเปิดการทำงาน
- close0 ปิดการทำงานทีซีพีซ็อกเก็ต
- wsaInit เริ่มต้นการทำงานซ็อกเก็ต

4.7 การทำงานกับจาร์ไฟล์ (JAR File Reader)

ประกอบไปด้วยไฟล์ jar.c , jar.h , inflate.c , inflate.h , inflateint.h , inflatetables.h ซึ่งมีรายละเอียดต่างๆ ดังนี้

4.7.1 jar.c , jar.h

จาร์ไฟล์(JAR File) นั้น เป็นการรวมเอาคลาสไฟล์หลายๆไฟล์มารวมกันให้เป็นไฟล์เดียว ช่วยให้เราสามารถแบ่งกลุ่มการทำงานของคลาสไฟล์ต่างๆ ได้ง่ายยิ่งขึ้น โดยหากเราต้องการใช้งานจาร์ไฟล์สามารถทำได้โดยตั้งค่า JAR_FILES_USE_STUDIO เป็น 1 (สามารถตั้งได้ที่ไฟล์ main.h)

อย่างไรก็ดี การนำจาร์ไฟล์มาใช้นั้นจะทำให้สิ้นเปลืองขนาดของหน่วยความจำมากกว่าปกติ ด้วยข้อจำกัดเรื่องของหน่วยความจำในอุปกรณ์ขนาดเล็กบางอุปกรณ์ ทำให้การนำจาร์ไฟล์มาใช้นั้น อาจจะมีหรือไม่มีก็ได้

การทำงานกับจาร์ไฟล์ฟังก์ชันที่สำคัญๆดังนี้

- openJARFile ทำการเปิดจาร์ไฟล์เพื่ออ่านข้อมูลภายใน ฟังก์ชันนี้จะมีการเรียกใช้ได้ 2 แบบด้วยกัน ขึ้นกับว่าจาร์ไฟล์นั้น จะใช้การอ่านจาก STDIO (JAR_FILES_USE_STDIO) หรือจาร์ไฟล์นั้นอยู่ในหน่วยความจำอยู่แล้ว
- closeJARFile ปิดการทำงานของจาร์ไฟล์ที่ได้ถูกเปิดโดยฟังก์ชัน openJarFile
- loadJARFileEntry อ่านข้อมูลของสมาชิกแต่ละตัวของจาร์ไฟล์ ผลที่ได้จากฟังก์ชันนี้คือการจองหน่วยความจำให้กับสมาชิกคลาสไฟล์ต่างๆ
- loadJARFileEntries อ่านข้อมูลของสมาชิกของจาร์ไฟล์ได้ทีละหลายๆไฟล์

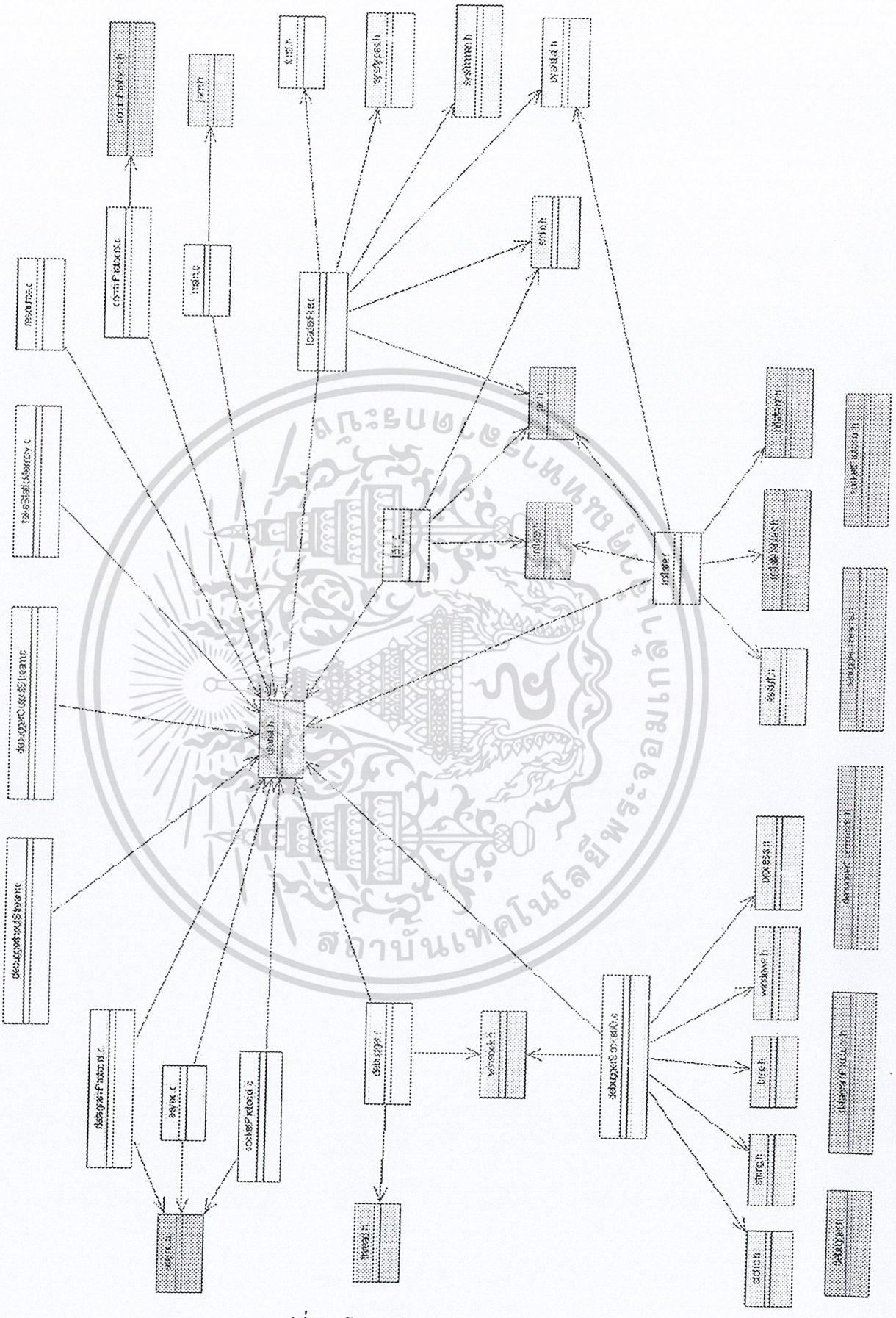
4.7.2 inflate.c , inflate.h , inflateint.h , inflatetables.h

จาร์ไฟล์(JAR File) นั้น เป็นการรวมเอาคลาสไฟล์หลายๆไฟล์มารวมกันให้เป็นไฟล์เดียว หรือการบีบอัดนั่นเอง การแปลงไฟล์จาร์ออกมาเป็นคลาสไฟล์ จะทำโดยขั้นตอนการ inflate นี้

การบีบอัดไฟล์และการรวมไฟล์นั้น จะเป็นรูปแบบเฉพาะของแต่ระระบบ ซึ่งจาร์ไฟล์นั้นก็มีรูปแบบเป็นของตัวเอง โดยไฟล์ inflate.c , inflate.h จะทำหน้าที่เป็นตัวหลักในการทำงานแปลความหมาย และเรียกการใช้งานจากไฟล์อื่นๆ inflateint.h จะทำหน้าที่กำหนดตัวแปรหรือค่าต่างๆที่จะใช้งาน (Initialize) ส่วนไฟล์ inflatetable.h นั้น จะเป็นตารางค่าที่ใช้ในการแปลความหมาย ซึ่งจะเก็บรูปแบบของการทำงานจาร์ไฟล์ไว้

การแปลความหมายของจาร์ไฟล์ฟังก์ชันที่สำคัญๆดังนี้

- inflateData ฟังก์ชันหลักในการแปลและขยายจาร์ไฟล์
- decodeDynamicHuffmanTables นำข้อมูลตารางจากไฟล์ inflatetables.h เพื่อใช้ในการแปลความหมายโค้ดต่างๆ
- makeCodeTable ทำการสร้างตารางกำหนดค่าต่างๆในตาราง



รูปที่ 4-1 โครงสร้างของ VmExtra

บทที่ 5
โครงสร้างและการทำงานในส่วนของ VmPort

จากโครงสร้างของเวอร์ชวลแมชีนขนาดเล็กนั้น ทุกสิ่งที่กล่าวมาจะไม่สามารถทำงานได้เลย หากไม่มีส่วนที่ใช้ติดต่อกับระบบปฏิบัติการนั้นๆ นั่นก็คือ VmPort นั่นเอง โดยในเวอร์ชวลแมชีนขนาดเล็กนั้น ได้แบ่งชื่อไปตามระบบปฏิบัติการที่เราจะนำมาใช้ เช่น ระบบปฏิบัติการวินโดว์ ก็จะใช้ชื่อว่า VmWin หรือระบบปฏิบัติการยูนิกซ์ ก็จะใช้ชื่อว่า VmUnix

5.1 machine_md.h

ทุกๆเวอร์ชวลแมชีนขนาดเล็กจะต้องมีไฟล์นี้ ไว้เพื่อตั้ง กำหนดค่าต่างๆที่จะใช้กับตัวอุปกรณ์ที่เราต้องการพัฒนา ไฟล์นี้จะอยู่ที่ "Vm*Port*/h/machine_md.h" โดยที่การตั้งค่าต่างๆในไฟล์นี้จะทำงานอยู่บนไฟล์ "main.h" อีกที นั่นคือหากเรากำหนดค่า ตัวแปรที่ซ้ำกับบนไฟล์ "main.h" โปรแกรมจะเลือกค่าที่ตั้งมาจาก "machine_md.h" ก่อน

ทุกสิ่งที่เกี่ยวข้องกับตัวอุปกรณ์โดยตรงจะกำหนดไว้ที่ไฟล์นี้ การกำหนดค่าต่างๆทั้ง typedef , #include , #dedfine รวมทั้งฟังก์ชันที่ทำงานติดต่ออุปกรณ์โดยตรงด้วย

ตัวแปรที่สำคัญๆดังนี้

- PATH_SEPARATOR กำหนดตัวแบ่งคลาสพาธเมื่อเรากำหนดคลาสพาธบนคอมมานไลน์โดยปกติจะเป็นตัว " : "
- กำหนดค่าต่างๆที่ใช้สำหรับระบุสถานะและไดเรกทอรีของฟังก์ชัน เช่น S_IFDIR เป็นสัญญานระบุไดเรกทอรี
- GetAndStoreNextKVMEvent เป็นค่าตัวแปรที่เปิดการทำงานที่จะรอการเรียกใช้งานจากภายนอกเข้ามาแล้วถึงจะเริ่มทำงาน ในทางปฏิบัติแล้วฟังก์ชันนี้จะช่วยให้อุปกรณ์ประหยัดการใช้แบตเตอรี่ของเครื่องได้
- กำหนดการทำงานเกี่ยวกับรูปภาพ(Graphic) (มีเฉพาะกับ VmWin)
- SLEEP_UNTIL ตั้งค่าให้หยุดเป็นเวลาที่กำหนดไว้ เมื่อถึงเวลาก็จะทำการปลุก(Wake Up) ให้กลับมาทำงานต่อไป

5.2 runtime_md.c

เป็นไฟล์ที่มีฟังก์ชันต่างๆเพื่อช่วยในการทำงานติดต่อกับอุปกรณ์ช่วยในการทำงานของเวอร์ชวลแมชีนขนาดเล็ก ฟังก์ชันที่สำคัญๆมีดังนี้

- AlertUser แสดงข้อความที่ทำงานผิดพลาดต่างๆออกมา
- allocateHeap ทำการร้องขอหน่วยความจำฮีป
- getpagesize ส่งค่าขนาดของหน้าจอที่กำหนดไว้
- allocateVirtualMemory_md ทำการร้องขอหน่วยความจำเสมือน (Virtual Memory)

- freeVirtualMemory_md ยกเลิกการจองหน่วยความจำเสมือน (Virtual Memory)
- protectVirtualMemory_md ทำการป้องกันการเรียกใช้หน่วยความจำเสมือน (Virtual Memory)
- CurrentTime_md รับค่าเวลาของระบบ(System Time) ตั้งแต่ปี 1970 หน่วยเป็นมิลลิวินาที
- InitializeNativeCode จะถูกเรียกตอนเริ่มการทำงานเพื่อเริ่มการกำหนดค่าเกี่ยวกับอุปกรณ์ต่างๆ
- nativeFinalization จะถูกเรียกตอนเริ่มการทำงานเพื่อจบการกำหนดค่าเกี่ยวกับอุปกรณ์ต่างๆ
- CallAsyncNativeFunction_md เรียกการทำงานอะซิงโครนัส ในทางปฏิบัติแล้วฟังก์ชันนี้แทบจะไม่ได้ทำงานเลยบนระบบปฏิบัติการวินโดว์ เพราะในที่สุดแล้วโปรแกรมที่ทำงานบนวินโดว์จะทำงานจนจบเธรดและจะทำจนกว่าจะไม่มีเธรดทำงานแล้ว

5.3 runtime_md2.c

ฟังก์ชันที่อยู่ในไฟล์นี้ถูกแบ่งการทำงานบางอย่างออกมาจาก "runtime_md.c" ทั้งนี้เพราะบางครั้งค่าตัวแปรที่ใช้งานอาจจะมีการสับสนกันได้ระหว่าง "windows.h" กับ "global.h"

ฟังก์ชันที่สำคัญๆมีดังนี้

- Calendar_md สร้างค่าเวลาในปัจจุบัน
- Yield_md จัดการยอมให้เธรดทำงานได้
- enterSystemCriticalSection ทำการรออยู่ใน มิวเท็กส์ของระบบ (System Mutex)
- exitSystemCriticalSection ออกจากมิวเท็กส์ของระบบ
- asyncThread ตัวหลักในการเริ่มทำงานของเธรด
- beginthreads เริ่มทำงานเธรด
- releaseAsyncThread หยุดการทำงานเธรด
- runtime2_md_init เป็นฟังก์ชันที่ให้เริ่มทำงานกำหนดค่าต่างๆให้กับระบบ
- sysTimeMillis ฟังก์ชันร้องขอค่าเวลาของระบบ

5.4 CommProtocol_md.c

กำหนดค่าและฟังก์ชันต่างๆเพื่อสนับสนุนให้เวอร์ชวลแมชีนขนาดเล็กสามารถทำงานผ่านพอร์ตซีเรียลได้ (Serial Port) ผ่านอุปกรณ์นั้นๆ

มีฟังก์ชันที่สำคัญๆดังนี้

- openPortByNumber , openPortByName ฟังก์ชันทำการเปิดพอร์ตโดยสามารถกำหนดได้ว่าจะเปิดพอร์ตใด
- configurePort ทำการกำหนดค่าต่างๆของพอร์ต ความเร็ว อัตราส่งข้อมูล เวลาการหยุดทำงาน (Timeouts)
- closePort ปิดการทำงานของพอร์ต
- writeToPort ส่งข้อมูลไปที่พอร์ต
- readFromPort อ่านข้อมูลที่ได้มาจากพอร์ต

- peekFromPort เข้าไปดูว่ามีการทำงานอะไรในพอร์ตหรือไม่
- getLastErrorMsg ส่งค่าผิดพลาดที่เกิดล่าสุดออกไป

5.5 datagramProtocol_md.c

กำหนดค่าและฟังก์ชันต่างๆเพื่อสนับสนุนให้เวอร์ชวลแมชีนขนาดเล็กสามารถทำงานดาต้าแกรมได้ (Datagram) ผ่านอุปกรณ์นั้นๆ โดยเรียกทำงานจากคลาสใน com.sun.cldc.io.j2me.datagram.Protocol

มีฟังก์ชันที่สำคัญๆดังนี้

- open0 เรียกใช้งานดาต้าแกรม
- send0 ทำการส่งข้อมูลดาต้าแกรม
- receive0 รับดาต้าแกรม
- close ปิดการทำงานดาต้าแกรม
- getMaximumLength รับค่าขนาดสูงสุดของดาต้าแกรม
- getNominalLength รับค่าขนาดของดาต้าแกรมตามปกติ
- getHostByAddr แปลงค่า IP Address มาเป็นชื่อของโฮสต์
- getIpNumber แปลงชื่อของโฮสต์มาเป็น IP Address

5.6 socketProtocol_md.c

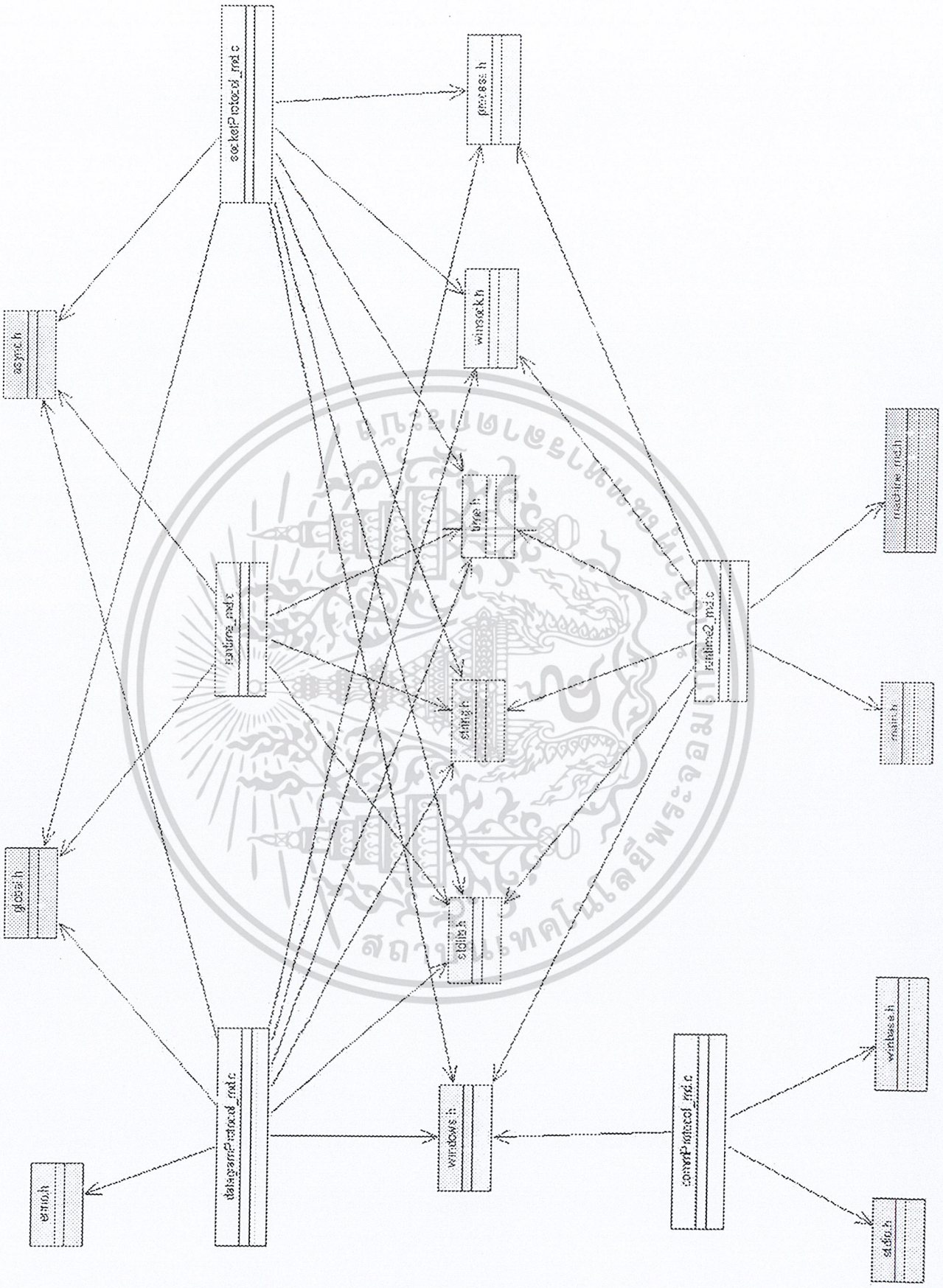
กำหนดค่าและฟังก์ชันต่างๆเพื่อสนับสนุนให้เวอร์ชวลแมชีนขนาดเล็กสามารถทำงานซ็อกเก็ต(socket) และเซิร์ฟเวอร์ซ็อกเก็ต (serversocket) ผ่านอุปกรณ์นั้นๆ โดยเรียกทำงานจากคลาสใน com.sun.cldc.io.j2me.socket.Protocol

มีฟังก์ชันที่สำคัญๆดังนี้

- setNonBlocking แปลงชื่อของโฮสต์มาเป็น IP Address
- getIpNumber แปลงค่า IP Address มาเป็นชื่อของโฮสต์
- open0 เปิดทีซีพีซ็อกเก็ต(TCP Socket)
- read0 อ่านข้อมูลจากทีซีพีซ็อกเก็ต
- write0 ส่งข้อมูลผ่านทีซีพีซ็อกเก็ต
- available0 ส่งค่าจำนวนไบต์ที่เหลืออยู่
- close0 ปิดการทำงานทีซีพีซ็อกเก็ต

ฟังก์ชันการทำงานของเซิร์ฟเวอร์

- open เปิดการทำงานผ่านทีซีพีซ็อกเก็ต
- accept รับและเริ่มเปิดการทำงาน
- close0 ปิดการทำงานทีซีพีซ็อกเก็ต
- wsaInit เริ่มต้นการทำงานซ็อกเก็ต



รูปที่ 5-1 โครงสร้างของ VmWin

บทที่ 6

การออกแบบและพัฒนา

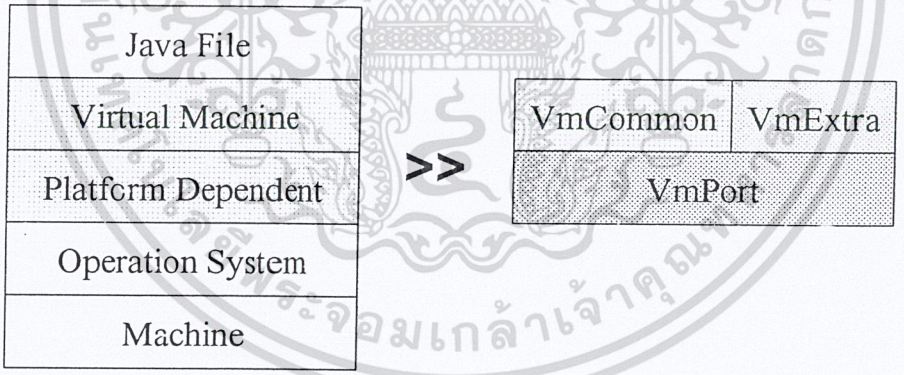
จากโครงสร้างของเวอร์ชวลแมชีนขนาดเล็กที่ได้อธิบายไปแล้วนั้น ในบทนี้จะเข้าสู่การพัฒนาเวอร์ชวลแมชีนขนาดเล็ก เพื่อให้สามารถใช้งานได้บนอุปกรณ์บอร์ดคอม 86 ซึ่งใช้ระบบปฏิบัติการเป็นดอส

6.1 ศึกษาโครงสร้างของเวอร์ชวลแมชีนขนาดเล็ก

โปรแกรมใดๆที่เขียนโดยภาษาจาวานั้น จะสามารถทำงานได้ในทุกๆระบบปฏิบัติการ เพียงระบบปฏิบัติการนั้นมีเวอร์ชวลแมชีน ถึงแม้ว่าทางบริษัทซันจะได้พัฒนาเวอร์ชวลแมชีนในหลายระบบปฏิบัติการ แต่ก็คงเป็นการยากที่จะสร้างเวอร์ชวลแมชีนได้บนทุกๆระบบปฏิบัติการ ทั้งนี้เพราะแต่ละระบบปฏิบัติการก็มีการพัฒนาตัวเองอยู่เสมอ

สำหรับเวอร์ชวลแมชีนขนาดเล็กก็เช่นกัน ทางบริษัทซันได้พัฒนาเวอร์ชวลแมชีนขนาดเล็กออกมาให้สามารถใช้งานได้กับระบบปฏิบัติการวินโดว์ (Windows) , ลีนุกซ์ (Linux) , โซลาลิส (Solaris) และ ยูนิกซ์ (Unix) เท่านั้น ทำให้เวอร์ชวลแมชีนขนาดเล็กไม่สามารถทำงานได้บนอุปกรณ์บอร์ดคอม 86 ที่มีดอสเป็นระบบปฏิบัติการ ทำให้เราจำเป็นที่จะต้องพัฒนาเวอร์ชวลแมชีนขนาดเล็กให้สามารถทำงานบนระบบปฏิบัติการดอสได้

เวอร์ชวลแมชีนขนาดเล็ก ในแต่ละระบบปฏิบัติการจะมีโครงสร้างดังรูป



รูปที่ 6-1 โครงสร้างของเวอร์ชวลแมชีนขนาดเล็ก

จากรูป ส่วนที่แลเงาคือ เวอร์ชวลแมชีน(Virtual Machine) กับ ส่วนที่ขึ้นอยู่กับแต่ละแพลตฟอร์ม (Platform Dependent) รวมกันเป็นเวอร์ชวลแมชีนขนาดเล็ก

- เวอร์ชวลแมชีน คือส่วนของการทำงานหลักเพื่อให้สามารถทำงานโปรแกรมภาษาจาวาได้ ซึ่งโค้ดของข้อมูลในส่วนนี้ก็คือ VmCommon และอาจจะมี VmExtra เข้ามาเพิ่มด้วยก็ได้

- ส่วนที่ขึ้นอยู่กับแต่ละแพลตฟอร์ม (Platform Dependent) นั้น ก็คือส่วนที่ติดต่อกับระบบปฏิบัติการ เนื่องจากแต่ละระบบปฏิบัติการจะมีการทำงานและการจัดการที่ต่างกันไป ซึ่งโค้ดในส่วนนี้ก็คือ VmPort

การที่จะพัฒนา VmPort สำหรับดอสขึ้นมาโดยเริ่มเขียนโปรแกรมขึ้นมาใหม่ทั้งหมดนั้น เป็นเรื่องที่ยากและเสียเวลาในการพัฒนาค่อนข้างมาก ดังนั้นการนำ VmPort ที่ได้มีการพัฒนามาแล้วมาดัดแปลงจึงเป็นเรื่องที่ง่ายและรวดเร็วกว่ามาก ซึ่งเราก็ต้องเลือกว่าจะนำ VmPort ตัวใดมาพัฒนาต่อ

VmPort สำหรับลีนุกซ์ และ ยูนิกซ์นั้น มีการทำงานที่ทำความเข้าใจได้ค่อนข้างซับซ้อนและยุ่งยาก เมื่อเทียบกับวินโดว์ โดยวินโดว์นั้นจะมีการทำงานในหลายส่วนที่คล้ายกับดอสมาก เพราะวินโดว์นั้นก็พัฒนาขึ้นมาจากพื้นฐานของดอสนั่นเอง ทำให้การพัฒนา VmPort จากวินโดว์มาเป็นดอสนั้นจะทำได้ง่ายกว่าลีนุกซ์ และ ยูนิกซ์

6.2 ข้อจำกัดของอุปกรณ์บอร์ดคอม86

บอร์ดคอม86 นั้นใช้ดอสเป็นระบบปฏิบัติการ โดยดอสที่ใช้นั้นมีหลากหลายเวอร์ชันด้วยกัน ซึ่งแต่ดอสแต่ละแบบนั้นก็จะมีความสามารถต่างๆกันไป แต่การทำงานโดยหลักๆนั้นยังคงเหมือนกัน เราจึงเลือกระบบปฏิบัติการดอสที่แถมมาด้วยกับตัวบอร์ด เพื่อให้ผู้ที่จะมาพัฒนาต่อหรือใช้งานต่อสามารถทำได้ง่าย

บอร์ดคอม86 มีความสามารถทำงานติดต่อผ่านเน็ตเวิร์กได้ ทำให้เราสามารถนำความสามารถทางด้านเน็ตเวิร์กของเวอร์ชวลแมชีนขนาดเล็กมาใช้ได้ โดยความสามารถทางด้านเน็ตเวิร์กนั้น จะอยู่ใน VmExtra และต้องมีการพัฒนาบางไฟล์บน VmPort ด้วยเช่นกัน

บอร์ดคอม86นั้น มีจำนวนหน่วยความจำที่ไม่มากนัก ทำให้การออกแบบต้องระวังเรื่องการเรียกใช้หน่วยความจำมากเป็นพิเศษ ทำให้การใช้งานบางอย่างเช่น การเรียกไฟล์นามสกุลจาร์(Jar File) จะทำได้ยากขึ้น เพราะการเรียกใช้ไฟล์จาร์จะใช้หน่วยความจำค่อนข้างมาก

ข้อเสียอย่างหนึ่งของระบบปฏิบัติการดอสคือไม่มีความสามารถทางด้านมัลติทาส์กิง (Multitasking) ซึ่งทำให้บอร์ดคอม86 ไม่สามารถใช้งานคำสั่งเธรด (Thread) ได้ ทำให้การพัฒนาเวอร์ชวลแมชีนขนาดเล็กบนดอสนั้น จะต้องตัดความสามารถทางด้านเธรดออกไป แต่การทำงานเธรดนั้นเป็นส่วนหลักของเวอร์ชวลแมชีนขนาดเล็ก ซึ่งมีการกำหนดไว้ใน VmCommon ทำให้เราตัดการทำงานนี้ออกไปไม่ได้ หรือหากจะตัดออกไปก็อาจจะทำให้การทำงานส่วนอื่นมีปัญหาไปด้วย ด้วยเหตุนี้ ผู้พัฒนาจึงยังคงการทำงานเธรดเอาไว้ในเวอร์ชวลแมชีนขนาดเล็ก แต่จะเป็นการกำหนดให้โปรแกรมภาษาจาวาที่จะทำงานบนเวอร์ชวลแมชีนขนาดเล็กจะต้องไม่เรียกใช้งานเธรดแทน

6.3 แผนงานการพัฒนา

จากที่เราได้ศึกษาโครงสร้างของเวอร์ชวลแมชีนขนาดเล็กและข้อจำกัดต่างๆของอุปกรณ์บอร์ดคอม86มาแล้ว ทำให้เราสามารถวางแผนงานการพัฒนาโครงสร้างเวอร์ชวลแมชีนขนาดเล็กบนบอร์ดคอม86 หรือบนระบบปฏิบัติการดอสได้ โดยการนำ VmPort ที่เป็นของระบบปฏิบัติการวินโดว์มาพัฒนา

โดยในช่วงแรกของการพัฒนานั้น จะเริ่มจากการตัดการทำงานของเวอร์ชวลแมชีนขนาดเล็กให้เหลือน้อยที่สุด โดยใช้งานเฉพาะที่จำเป็นเท่านั้น แล้วทดลองทำงานบนบอร์ดคอม86 ดูว่าใช้งานได้หรือไม่ และแก้ไขพัฒนาให้สามารถใช้งานได้ต่อไป จากนั้นจึงค่อยๆเพิ่มความสามารถ ฟังก์ชันต่างๆเข้าไปทีละนิด และทำงานเป็นเวอร์ชันต่อๆไป จนถึงเวอร์ชันที่สมบูรณ์เสร็จสิ้น

เวอร์ชัน	รายละเอียด
0.1	ตัดส่วนการทำงานอื่นๆออกหมด เหลือเพียงความสามารถในการแสดงเวอร์ชันการทำงานออกหน้าจอ
0.3	สามารถกำหนดไดเรกทอรีของคลาสได้ (Class Path)
0.5	สามารถกำหนดการเรียกใช้ การจัดการหน่วยความจำได้ (Allocate Memory) สามารถกำหนดการเรียกขอหน่วยความจำฮีปได้ (Initial Heap Size)
0.7	สามารถโหลดคลาสไฟล์มาใช้งานได้ (Load Class File) สามารถแยกประเภทของไฟล์ต่างๆได้ว่าเป็นประเภทคลาสไฟล์ ไดเรกทอรี หรือ จาร์ไฟล์ได้
0.9	สามารถทำงานกับคลาสไฟล์ที่เรียกใช้งานร่วมกับคลาสอื่นได้
1.0	ทำงานได้เช่นเดียวกับเวอร์ชวลแมชีนขนาดเล็ก แต่ตัดการทำงานเธรด และเน็ตเวิร์กออก

ตารางที่ 6-1 แผนงานการพัฒนาเวอร์ชวลแมชีนขนาดเล็กสำหรับบอร์ดคอม86

6.4 การทดลองใช้งานเวอร์ชวลแมชีนขนาดเล็ก

เวอร์ชวลแมชีนขนาดเล็กที่ทดลองใช้งานและทำการพัฒนาคือเวอร์ชัน 1.0.4

6.4.1 เริ่มต้นทดลองใช้งานเวอร์ชวลแมชีนขนาดเล็ก

การใช้งาน J2ME CLDC เวอร์ชัน 1.04 นั้น เมื่อเราทำการติดตั้งโปรแกรมลงบนคอมพิวเตอร์แล้ว เราสามารถเรียกใช้งานเวอร์ชวลแมชีนขนาดเล็กที่ทำการคอมไพล์มาพร้อมใช้งานได้เลย ซึ่งไฟล์ดังกล่าวจะอยู่ในไดเรกทอรี "/Bin" ซึ่งจะมีการแยกตามแพลตฟอร์มต่างๆ โดยเราจะทดลองใช้งานแพลตฟอร์มวินโดว์ที่เราจะนำมาพัฒนา โดยเข้าไปที่ไดเรกทอรี "/win32" จะมีไฟล์ให้เราสามารถใช้งานได้ 4 ไฟล์ด้วยกัน ดังนี้

- kvm.exe เป็นไฟล์หลักที่เรียกใช้งานเวอร์ชวลแมชีนขนาดเล็ก สำหรับวิธีใช้งานนั้น เราสามารถทำงานผ่านทางคอมมานพรอมต์ (Command Prompt) และกำหนดคำสั่งต่างๆได้ มีตัวอย่างการใช้งานดังนี้

คำสั่ง	รายละเอียด
Kvm	แสดงวิธีการเรียกใช้งานของโปรแกรมเวอร์ชวลแมชีนขนาดเล็ก
Kvm <classfile>	เป็นการเรียกใช้งานคลาสไฟล์ เช่น " kvm HelloWorld " ตัวอย่างนี้คือการเรียกไฟล์ HelloWorld.class มาใช้งาน
Kvm –version	แสดงเวอร์ชันที่เราใช้งานเวอร์ชวลแมชีนขนาดเล็กอยู่ ซึ่งตัวเวอร์ชันนี้จะเปลี่ยนไปตามที่เรากำหนดในโปรแกรม
Kvm –classpath <path>	เป็นการกำหนดคลาสพาธที่เราใช้ ในกรณีที่คลาสไฟล์ที่เราต้องการเรียกใช้งานนั้นอยู่คนละไดเรกทอรีที่เวอร์ชวลแมชีนขนาดเล็กอยู่ เช่น "kvm –classpath c:/class HelloWorld " ตัวอย่างนี้คือการเรียกไฟล์ HelloWorld.class ซึ่งอยู่ในไดเรกทอรี c:/class
Kvm –heapsize <size>	เป็นการกำหนดขนาดของฮีป หรือขนาดของหน่วยความจำสำรองให้กับการทำงานของ

	เวอร์ชวลแมชีนขนาดเล็ก เนื่องจากเวอร์ชวลแมชีนขนาดเล็กนั้นจะกำหนดขนาดของฮีปเอาไว้ที่ 65532 ไบท์ (bytes) หากโปรแกรมที่เราต้องการเรียกใช้ต้องมีการจองหน่วยความจำที่มากหรือน้อยกว่านั้น เราสามารถกำหนดได้โดยคำสั่งนี้ เช่น "kvm –heapsize 128k HelloWorld " ตัวอย่างนี้คือการเรียกไฟล์ HelloWorld.class มาใช้งาน โดยกำหนดขนาดของหน่วยความจำที่จะมาใช้งานเท่ากับ 128 กิโลไบท์ (128 Kilo Bytes)

ตารางที่ 6-2 ตัวอย่างการใช้งานเวอร์ชวลแมชีนขนาดเล็ก

- kvm_j.exe เป็นการเรียกใช้งานเวอร์ชวลแมชีนขนาดเล็กโดยมีการใช้ตัวควบคุมการทำงานของจาวา(Java Apllication Manager หรือ JAM) ซึ่งได้อธิบายความสามารถของ JAM ไปแล้วก่อนหน้านี้ สำหรับการเรียกใช้งานนั้นจะเหมือนกับ kvm.exe

- preverify.exe คือโปรแกรมที่ใช้ตรวจสอบความถูกต้องของคลาสไฟล์ว่ามีการเขียนคำสั่งต่างๆถูกต้องหรือไม่ โดยมีวิธีการเรียกใช้งานคือ preverify <classfile> เช่น "preverify HelloWorld "

- kvm_g.exe ไฟล์นี้จะอยู่ในไดเรกทอรี "/debug " เมื่อเรียกใช้งาน นอกจากจะทำงานเหมือนกับเวอร์ชวลแมชีนขนาดเล็กแล้ว ยังทำการดีบั๊กโดยการแสดงการทำงานส่วนต่างๆแทรกอยู่ เพื่อให้เห็นการทำงานที่ชัดเจนมากขึ้น โดยมีการทำการต่างๆดังนี้ (แสดงเฉพาะคำอธิบายที่ตามด้วย kvm_g)

คำสั่ง	รายละเอียด
-traceallocation	แสดงการจองหน่วยความจำ
-tracedebugger	แสดงการทำดีบั๊กเกอร์
-traceegc	แสดงการทำ Garbage Collection
-traceegcverbose	แสดงการทำ Garbage Collection ที่ละเอียดมากขึ้น
-traceclassloading	แสดงการโหลดคลาสต่างๆขึ้นมา
-traceclassloadingverbose	แสดงการโหลดคลาสต่างๆขึ้นมา ที่ละเอียดมากขึ้น
-traceverifier	แสดงการตรวจสอบคลาสไฟล์
-tracestackmaps	แสดงการทำ Stack Maps
-tracebytecodes	แสดงการทำงานระดับของ Bytecode
-tracemethods	แสดงการทำงานของเมธอดต่างๆ
-tracemethodsverbose	แสดงการทำงานของเมธอดต่างๆที่ละเอียดมากขึ้น
-traceframes	แสดงการทำ Stack Frames
-tracestackchucks	แสดงการร้องขอหน่วยความจำเพื่อทำ Stack Chucks ตัวใหม่
-traceexceptions	แสดงการทำงานที่ผิดปกติ
-traceevents	แสดงการทำงานของระบบ event
-tracethreading	แสดงการทำงานของระบบเธรด

-tracemonitor	แสดงการทำงานของ monitor objects
-tracenetworking	แสดงการทำงานของระบบเน็ตเวิร์ค
-traceall	แสดงการทำงานทุกๆอย่าง

ตารางที่ 6-3ตัวอย่างการใช้งานเวอร์ชวลแมชีนขนาดเล็กที่มีดีบักด้วย

6.4.2 วิธีการคอมไพล์โค้ดเวอร์ชวลแมชีนขนาดเล็ก

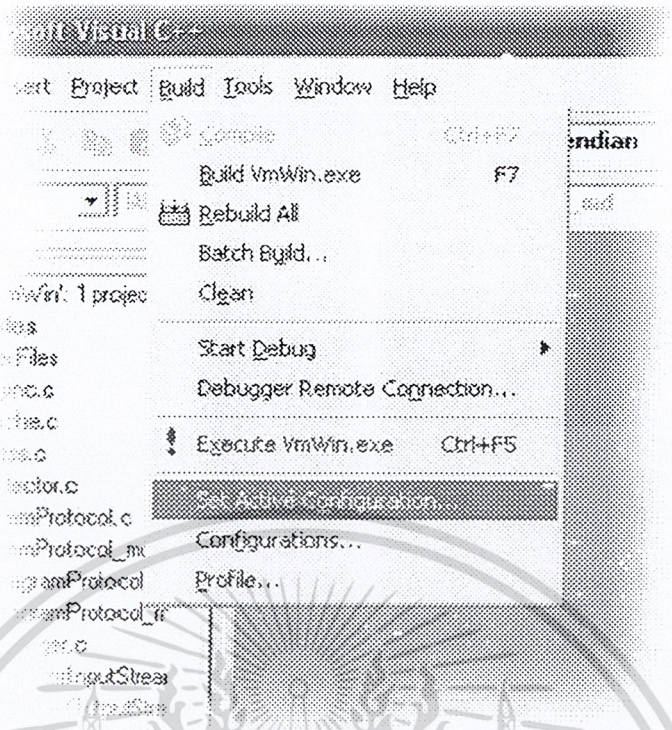
เวอร์ชวลแมชีนขนาดเล็กนั้น เขียนขึ้นมาด้วยโค้ดภาษาซี โดยมีการแบ่งไฟล์ออกเป็นหลายๆไฟล์แยกตามการทำงาน และการประกาศค่าต่างๆไว้ ทำให้การคอมไพล์โค้ดเวอร์ชวลแมชีนขนาดเล็กนั้น จำเป็นที่จะต้องทำการคอมไพล์ไฟล์ร่วมกันหลายไฟล์ ซึ่งมีวิธีการคอมไพล์ได้หลายวิธีดังนี้

- GNU C , GNU Make เป็นวิธีที่สามารถทำได้ทั้งบนระบบปฏิบัติการยูนิกซ์ และลีนุกซ์ โดยเข้าไปที่ไดเรกทอรี "/build/linux " แล้วเรียกคำสั่ง "gnumake " โปรแกรมจะทำการคอมไพล์โค้ดภาษาซีให้ และทำการรวบรวมไฟล์ต่างๆ รวบรวมเข้ามาเป็นไฟล์ kvm.exe ซึ่งเรายังสามารถกำหนดข้อกำหนดต่างๆได้จากไฟล์ "Makefile "

- GNU C , GNU Make (บนวินโดว์) เป็นวิธีที่สามารถทำได้ทั้งบนระบบปฏิบัติการวินโดว์ แต่เราจะต้องมีโปรแกรม GNU C เวอร์ชัน 2.95.2 ขึ้นไป จากนั้นเข้าไปที่ไดเรกทอรี "/build/win32 " แล้วเรียกคำสั่ง "gnumake " โปรแกรมจะทำการคอมไพล์โค้ดภาษาซีให้ และทำการรวบรวมไฟล์ต่างๆรวบรวมเข้ามาเป็นไฟล์ kvm.exe ซึ่งเรายังสามารถกำหนดข้อกำหนดต่างๆได้จากไฟล์ "Makefile " แต่วิธีนี้มักจะมีปัญหาที่เราต้องแก้ไขในโค้ดของ makefile บ้าง เพราะคำสั่ง make นั้น เป็นคำสั่งที่สร้างขึ้นมาจากระบบปฏิบัติการยูนิกซ์ ทำให้คำสั่งหลายๆอย่างจะมีปัญหากับการทำงานบนวินโดว์

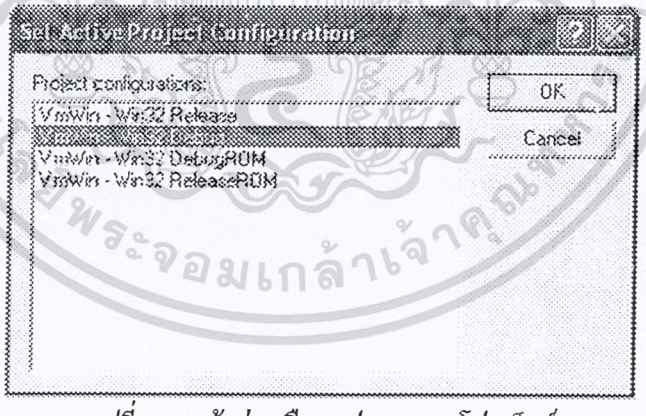
- Microsoft Visual C++ 6.0 Prefessional เป็นวิธีที่ง่ายและสะดวกที่สุด ทั้งนี้เนื่องจากการ ใช้คำสั่ง make นั้น บางครั้งต้องมีการกำหนดค่าต่างๆอีก ถึงจะสามารถ ใช้งานได้ แต่การคอมไพล์เวอร์ชวลแมชีนขนาดเล็กด้วย Microsoft Visual C++ นั้น เพียงเราเข้าไปในไฟล์โปรเจ็กต์ ที่อยู่ใน ไดเรกทอรี "/kvm/VmWin/build/VmWin.dsw "

นอกจากที่เราจะสามารถคอมไพล์และสร้าง ไฟล์เวอร์ชวลแมชีนขนาดเล็กได้แล้ว การสร้างเวอร์ชวลแมชีนขนาดเล็กผ่าน Visual C++ นั้น เรายังสามารถกำหนดค่าต่างๆได้ง่ายกว่าวิธีอื่นมาก ดังรูปเป็นตัวอย่างการกำหนดรูปแบบที่จะสร้างเวอร์ชวลแมชีนขนาดเล็กออกมาได้อีกด้วย ดังรูป



รูปที่ 6-2 การตั้งค่าในการสร้างเวอร์ชวลแมชีนขนาดเล็ก

จากรูป เราสามารถที่จะตั้งค่าโปรเจ็กต์ที่จะออกมาโดยไปเลือกที่ทูลบาร์ เลือกหัวข้อ “Build” จากนั้นเลือก “Set Active Configuration”



รูปที่ 6-3 หน้าต่างเลือกรูปแบบของโปรเจ็กต์

จากรูป คือชื่อของโปรเจ็กต์ที่จะออกมา โดยเราสามารถเลือกให้เวอร์ชวลแมชีนขนาดเล็กที่คอมไพล์ออกมาเป็นรูปแบบที่เราต้องการได้ โดยมีรายละเอียดดังนี้

ชื่อรูปแบบ	รายละเอียด
Win32 Release	จะได้เวอร์ชวลแมชีนขนาดเล็กปกติ

Win32 Debug	จะได้เวอร์ชวลแมชีนขนาดเล็กที่ทำการดีบั๊กได้
Win32 ReleaseROM	จะได้เวอร์ชวลแมชีนขนาดเล็กที่มีการใช้รอม(Romizing) (จากหัวข้อ 7.4.1 คือ kvm.exe)
Win32 DebugROM	จะได้เวอร์ชวลแมชีนขนาดเล็กที่มีการใช้รอมและสามารถทำการดีบั๊กได้ (จากหัวข้อ 7.4.1 คือ kvm_g.exe)

ตารางที่ 6-4 รูปแบบเวอร์ชวลแมชีนขนาดเล็กที่ทำบน Microsoft Visual C++

6.5 การพัฒนาและการแก้ไขโค้ดต่างๆ

หลังจากที่ได้ศึกษาโครงสร้างของเวอร์ชวลแมชีนขนาดเล็ก วางแผนการพัฒนาและทดลองใช้งานดูแล้ว การพัฒนาและแก้ไขโค้ดต่างๆของเวอร์ชวลแมชีนขนาดเล็กให้สามารถทำงานได้บนอุปกรณ์บอร์ดคอม 86ต่อไป ซึ่งเราจะนำเอา VmWin ซึ่งใช้เป็นตัวกลางในการติดต่อระหว่างเวอร์ชวลแมชีนขนาดเล็กกับวินโดว์มาพัฒนา

6.5.1 ความแตกต่างระหว่างตัวแปรวินโดว์และดอส

ระบบปฏิบัติการวินโดว์และดอสนั้นมีความแตกต่างกันพอสมควร วินโดว์นั้นทำงานอยู่ที่ระบบ 32 บิต ส่วนดอสนั้นจะทำงานอยู่ที่ 16 บิต ดังนั้นค่าตัวแปรต่างๆที่กำหนดในวินโดว์อาจจะไม่สามารถทำงานบนดอสได้ นอกจากนั้นการที่ดอสนั้น ทำงานผ่านคอมมานด์ไลน์ ทำให้ดอสเองไม่มีส่วนติดต่อกับผู้ใช้ (User Interface) ซึ่งต่างจากวินโดว์ ที่จะมีการทำงานส่วนติดต่อกับผู้ใช้ที่มากกว่า

การกำหนดชนิดของข้อมูล(data types) ของวินโดว์นั้น มีส่วนที่ต่างกับดอสอยู่พอสมควรดังตาราง

วินโดว์	ดอส (หรือบนภาษา C)	อธิบาย
BYTE	unsigned char	เป็น 8 บิตเสมอ
WORD	unsigned short	เป็น 16 บิตเสมอ
int	int	เป็น 16 หรือ 32 บิตขึ้นอยู่กับแต่ละแพลตฟอร์ม
UINT	unsigned int	เป็น 16 หรือ 32 บิตขึ้นอยู่กับแต่ละแพลตฟอร์ม
LONG	long	ตัวแปรชนิด Long (อาจจะเป็น 64 บิตในบางแพลตฟอร์ม)
ULONG	unsigned long long	ตัวแปรชนิด Long (อาจจะเป็น 64 บิตในบางแพลตฟอร์ม) เป็น 32 บิตเสมอ
DWORD	unsigned long	เป็น 32 บิตเสมอ
BOOL	int	ในภาษา C++ อาจจะมีการใช้ enum แทน
TRUE	1	เป็นจริง
FALSE	0	เป็นเท็จ

ตารางที่ 6-5 เปรียบเทียบการประกาศตัวแปรต่างๆบนวินโดว์กับในภาษาซี

ซึ่งในไฟล์ต่างบน VmWin นั้น จะมีการประกาศตัวแปรต่างๆที่ต่างกับบนดอสหรือภาษาซี เราจึงต้องทำการแปลงค่าตัวแปรให้เป็นไปตามที่ภาษาซีกำหนดดังนี้

- #define FALSE 0
- #define TRUE 1
- typedef long Long;
- typedef char Boolean;
- typedef long long64;
- typedef unsigned long ulong64;
- typedef long jlong;
- typedef unsigned char BYTE;
- typedef int BOOL;

6.5.2 ไฟล์ส่วนกลางต่างๆ

การทำงานของวินโดว์จะมีการเรียกใช้อินคลูดไฟล์(Include File)และไลบรารีต่างๆ เช่น “Windows.h” , “Winbase.h” , “winsock.h” , “winsock2.h” , “process.h” ซึ่งจะใช้งานได้เฉพาะกับวินโดว์เท่านั้น ทำให้เราต้องนำอินคลูดไฟล์ที่สามารถใช้งานได้บนดอสมาดัดแปลง เช่น “dos.h” เป็นต้น

การพัฒนาในส่วนนี้ ทำได้โดยการที่เราตัดการเรียกไฟล์ที่เกี่ยวข้องกับวินโดว์ออกให้หมด และใช้คอมไพเลอร์บอแลนด์ซี เวอร์ชัน 3 (Borland C 3.0)ที่สามารถเขียนโปรแกรมแล้วนำไปใช้บนดอสได้ดี มาใช้ทำงาน และใช้อินคลูดไฟล์จากบอแลนด์แทน

ตัวอย่างการแก้ไขคือในไฟล์ “runtime2_md.c” นั้น จะมีฟังก์ชันที่ทำงานเรื่องเวลาคือ

```
Calendar_md(void)
{
  SYSTEMTIME st;
  //initialize
  memset(&st, 0, sizeof(st));
  memset(&date, 0, sizeof(date));
  GetSystemTime(&st);
  // initialize calendar fields
  date[YEAR] = st.wYear;
  date[MONTH] = st.wMonth;
  date[DAY_OF_MONTH] = st.wDay;
  date[HOUR] = st.wHour;
  date[MINUTE] = st.wMinute;
  date[SECOND] = st.wSecond;
  date[MILLISECOND] = st.wMilliseconds;
```

```
    return date;
}
```

จากฟังก์ชันนี้ จะมีการเรียกใช้ฟังก์ชัน Systemtime และ Getsystemtime ซึ่งเป็นการเรียกค่าเวลาของเครื่องเพื่อนำมาใช้งาน จะอยู่ในไฟล์ "windows.h" ทำให้เมื่อนำมาใช้กับดอสแล้วจะมีปัญหา เราจึงต้องแปลงโดยใช้วิธีนำฟังก์ชันการดึงเวลาจาก "time.h" บนบอแลนด์ซี 3 มาใช้ ดังนี้

```
Calendar_md(void)
{
 clock_t clk;
   struct tm *tmP = NULL;
   /* initialize the date array */
   memset(&date, 0, MAXCALENDARFLDS);
   /* returns the time */
   time(&clk);
   /* returns pointer to tm struct */
   tmP = localtime(&clk);
   /* initialize the calendar fields */
   date[YEAR] = tmP->tm_year;
   date[MONTH] = tmP->tm_mon;
   date[DAY_OF_MONTH] = tmP->tm_mday;
   date[HOUR] = tmP->tm_hour;
   date[MINUTE] = tmP->tm_min;
   date[SECOND] = tmP->tm_sec;
   /* We cannot accurately determine this value,
    * so set it to 0 for now
    */
   date[MILLISECOND] = 0;
   return date;
}
```

6.5.3 ความผิดพลาดบนคอมไพเลอร์

เนื่องจากคอมไพเลอร์ที่เหมาะกับการเขียนโปรแกรมบนบอร์ดคอม 86 มากที่สุดนั้นคือบอแลนด์เวอร์ชัน 3 ซึ่งเป็นคอมไพเลอร์ที่ถูกพัฒนามานาน และไม่สนับสนุนการเขียนโปรแกรมในแบบใหม่หลายรูปแบบ ซึ่งเวอร์ชวลแมชีนขนาดเล็กที่เรานำมาใช้นั้น มีการเขียนโค้ดข้อมูลในรูปแบบที่ทันสมัย และซับซ้อน ทำให้การทำงานบางอย่างคอมไพเลอร์จะเข้าใจในโค้ดที่มีอยู่

หากได้ลองนำโค้ดเวอร์ชวลแมชีนขนาดเล็กเวอร์ชัน 1.0.4 มาลองคอมไพล์ดูจะพบกับข้อผิดพลาดมากมาย ทำให้เราต้องแก้ไขโค้ดให้สามารถทำงานผ่านการคอมไพล์ให้ได้ โดยปัญหาที่พบโดยทั่วไปคือ

- การเรียกฟังก์ชันที่อยู่นอกไฟล์ โดยปกติแล้วหากเราต้องการที่จะเรียกฟังก์ชันที่อยู่นอกไฟล์และไม่ได้อยู่ในอินคลูดไฟล์นั้น จะต้องประกาศฟังก์ชันนั้นว่าเป็นฟังก์ชันที่อยู่นอกไฟล์ โดยคำสั่ง “extern” เช่น extern int HelloWorld(); ในโค้ดบางฟังก์ชันของเวอร์ชวลแมชีนขนาดเล็กนั้น ไม่ได้มีการกำหนด “extern” เอาไว้ เราจึงต้องทำการประกาศให้คอมไพเลอร์ทราบ
- การสร้างโปรเจ็กต์บนบอแลนด์ซีเวอร์ชัน 3 นั้น เราจะต้องกำหนดค่าต่างๆให้สามารถทำงานบนดอสได้ มิเช่นนั้นโปรแกรมที่คอมไพล์ออกมาแล้ว จะมีปัญหาเกิดขึ้น โดยสิ่งที่เราต้องกำหนดในการสร้างโปรเจ็กต์นั้น คือไปที่ทูลบาร์ Option -> Compiler ->Entry/Exit Code Generation จากนั้นเลือกให้โปรแกรมที่ออกมาเป็นดอส

  นอกจากนั้น เราต้องกำหนดขนาดของโปรเจ็กต์ ซึ่งโดยปกติแล้วคอมไพเลอร์จะตั้งให้ขนาดของโปรเจ็กต์เป็นขนาดเล็ก (Smail) ซึ่งเวอร์ชวลแมชีนขนาดเล็กนั้น มีขนาดของโปรเจ็กต์ที่ใหญ่ เราจึงควรแก้ไขขนาดของโปรเจ็กต์ ซึ่งทำได้โดยไปกำหนดที่ทูลบาร์ Option -> Compiler -> Code Generation Options

6.5.4 การจำกัดขนาดของเวอร์ชวลแมชีนขนาดเล็ก

เนื่องด้วยข้อจำกัดเรื่องหน่วยความจำและเนื้อที่เก็บข้อมูลของบอร์ดคอม 86 นั้นมีไม่มากนัก ทำให้เราต้องระวังถึงขนาดของโปรแกรมที่ทำการคอมไพล์ออกมาแล้ว ไม่ให้มีขนาดเกินที่บอร์ดคอม 86 จะรองรับได้

โดยปกติแล้วบอร์ดคอม 86 นั้นจะมีเนื้อที่เก็บข้อมูลประมาณ 512 กิโลไบต์ ซึ่งจะต้องแบ่งให้กับข้อมูลของระบบปฏิบัติการ ซึ่งก็คือดอสนั่นเอง โดยจะเหลือเนื้อที่เหลือให้สามารถนำเวอร์ชวลแมชีนขนาดเล็กลงไปทำงานได้ประมาณ 256 กิโลไบต์เท่านั้น

ขนาดของเวอร์ชวลแมชีนขนาดเล็กที่เวอร์ชัน 1.0.4 โดยตั้งให้สามารถทำงานได้ทุกอย่าง บนระบบปฏิบัติการวินโดว์จะมีขนาดประมาณ 264 กิโลไบต์ โดยหากต้องการทำดีบั๊กเข้าไปด้วยนั้น จะมีขนาดเพิ่มขึ้นเป็นประมาณ 557 กิโลไบต์ ซึ่งจะเห็นได้ว่ามีขนาดเกินกว่าเนื้อที่ว่างบนบอร์ดคอม 86 ทำให้เราต้องพยายามลดขนาดของเวอร์ชวลแมชีนขนาดเล็กลง

การลดขนาดของเวอร์ชวลแมชีนขนาดเล็กนั้น เป็นสิ่งที่ทำได้ยากมาก ถึงแม้เราจะพยายามลดฟังก์ชันการทำงานลงก็ตาม เพราะโปรแกรมที่ถูกคอมไพล์บนวินโดว์นั้น จะมีขนาดที่ไม่มากนัก เพราะเวอร์ชวลแมชีนขนาดเล็กสามารถเรียกใช้งานฟังก์ชัน ไลบรารีต่างๆที่มีบนวินโดว์มาใช้งานได้เลย แต่การจะคอมไพล์แล้วนำมาใช้บนดอสนั้น คำสั่งบางคำสั่งจะต้องใช้การทำงานที่มากกว่าบนวินโดว์มาก การลดขนาดจึงเป็นเรื่องยากพอสมควร

6.5.5 การแก้ไขที่นอกเหนือจาก VmPort

จากโครงสร้างของเวอร์ชวลแมชีนขนาดเล็ก ไฟล์ที่อยู่ใน VmCommon และ VmExtra นั้น เมื่อเราต้องการจะนำมาใช้ในอุปกรณ์ใดๆ เราไม่จำเป็นที่จะต้องเปลี่ยนแปลงการทำงานในนั้นเลย เราเพียงเปลี่ยนแปลงโค้ดใน VmPort ก็เพียงพอ แต่จากข้อมูลที่พบจากผู้พัฒนารายอื่นๆ และจากการทดลองใช้งานดูเองแล้ว จะพบว่า เราจำเป็นที่จะต้องแก้ไขการทำงานบางอย่างนอกจากใน VmPort อีกด้วย ทั้งในส่วนของ VmCommon และ VmExtra ด้วยเช่นเดียวกัน

ซึ่งปัญหานี้เป็นปัญหาที่ใหญ่มาก เพราะเราเองก็ไม่ทราบว่าโค้ดส่วนใดที่เราจะต้องไปแก้ไขการทำงาน โค้ดส่วนใดที่ไม่ต้องแก้ไข ส่วนที่แก้ไขไปแล้วจะทำให้การทำงานผิดพลาดไปด้วยหรือไม่ ซึ่งเปรียบได้เหมือนการงมเข็มในมหาสมุทรเลยก็ว่าได้

วิธีการแก้ปัญหานี้นั้น เราจะต้องทำการดีบั๊กการทำงานของเวอร์ชวลแมชีนขนาดเล็กของเราที่คอมไพล์มาได้กับเวอร์ชวลแมชีนขนาดเล็กที่สมบูรณ์แล้ว เปรียบเทียบการทำงานไปทีละขั้นว่ามีส่วนใดที่ได้ผลการทำงานต่างจากการทำงานจริง โดยส่วนมากแล้วข้อผิดพลาดมักจะมาจากสาเหตุที่คอมไพเลอร์ ดังที่ได้อธิบายไปในหัวข้อก่อนหน้านี้แล้ว

ตัวอย่างปัญหาใน VmCommon

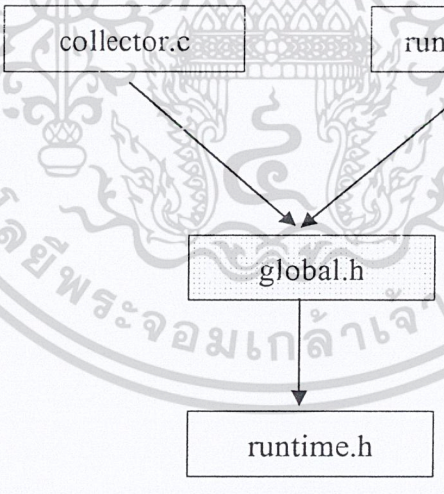
ไฟล์ collector.c

```
AllHeapStart = allocateHeap(&VMHeapSize, &TheHeap);
```

ไฟล์ runtime_md.c

```
cell *allocateHeap(long *sizeptr, void **realresultptr) {
    void *space = malloc(*sizeptr + sizeof(cell) - 1);
    *realresultptr = space;
    return (void *) ((((long)space) + (sizeof(cell) - 1)) & ~(sizeof(cell) - 1));
}
```

ฟังก์ชัน allocateHeap นั้นจะทำการร้องขอหน่วยความจำเพื่อใช้สำหรับเป็นฮีปในเวอร์ชวลแมชีนขนาดเล็ก โดยโครงสร้างการของไฟล์ทั้ง 2 เป็นดังนี้



รูปที่ 6-1 โครงสร้างของไฟล์ collector.c และ runtime_md.c

จากรูป จะเห็นว่าการที่ไฟล์ collector.c เรียกใช้งานฟังก์ชันจากใน runtime_md.c โดยที่จะมีการประกาศชื่อฟังก์ชันอีกครั้งที่ไฟล์ runtime.h ซึ่งถ้าพิจารณาตามหลักการเขียนโปรแกรมภาษาซีนั้น ก็ถือว่าถูกต้อง แต่เมื่อเรานำไปใช้งานในคอมไพเลอร์ที่เราเลือกใช้สำหรับดอส (Borland C 3.0) จะได้ค่าของพอยเตอร์ที่ส่งกลับมาผิดพลาด ซึ่งอาจจะเป็นเพราะลักษณะของการเขียนโปรแกรมที่ไม่เหมาะกับคอมไพเลอร์สมัยเก่า

จากปัญหาดังกล่าว เราจึงการแก้ไขด้วยการนำฟังก์ชันร้องขอหน่วยความจำ หรือ malloc มาไว้ที่ไฟล์ collector.c เองเลย แล้วส่งค่าพอยเตอร์ที่ชี้ไปที่ไฟล์ runtime_md.c เอง ทั้งนี้การแก้ปัญหานี้อาจจะแก้ได้ด้วยวิธีอื่นๆก็ได้ แล้วแต่ความถนัดของแต่ละคน

โค้ดจากที่ได้แก้ไขแล้ว

ไฟล์ collector.c

```
AllHeapStart = malloc(VMHeapSize);
allocateHeap(&AllHeapStart,&TheHeap);
```

ไฟล์ runtime_md.c

```
void allocateHeap(long *space, void **realresultptr){
    *realresultptr = space;
}
```

บทที่ 7

ผลการทดลอง

การใช้งานเวอร์ชวลแมชีนขนาดเล็กที่ได้พัฒนาให้ใช้บนอุปกรณ์บอร์ดคอม86นั้น สามารถใช้งานได้ดังนี้

7.1 การใช้งานผ่านระบบปฏิบัติการดอสบนคอมพิวเตอร์

การทำงานผ่านระบบปฏิบัติการดอสนั้น หลายคนจะเข้าใจว่า เราสามารถทำงานได้โดยใช้โปรแกรมคอสพรอมต์(Dos Prompt) ที่มีในระบบปฏิบัติการวินโดว์ทั่วไป ซึ่งโปรแกรมดอสพรอมต์นั้นจะทำงานจำลองการทำงานบนดอส ทำงานผ่านทางคอมมานพรอมต์(Command Prompt) แต่โปรแกรมดอสพรอมต์นั้น จะเรียกการใช้งานพลายอย่างจากวินโดว์ เช่น การจัดการเรื่องหน่วยความจำ

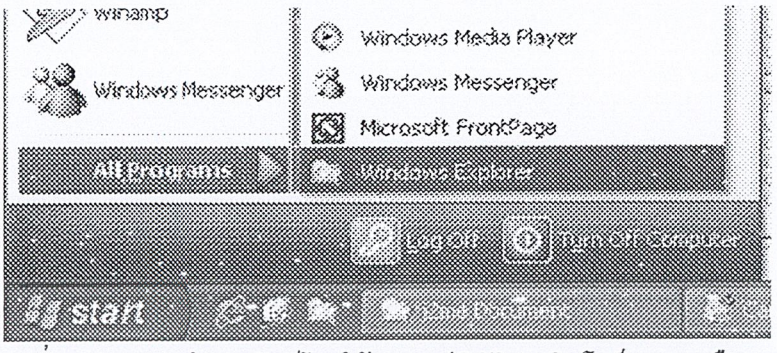
ดังนั้น การทดลองทำงานเวอร์ชวลแมชีนขนาดเล็กบนคอมพิวเตอร์นั้น ควรจะใช้เครื่องที่มีดอสเป็นระบบปฏิบัติการจริงๆ หรือทำโดยการสร้างแผ่นเปิดเครื่องหรือบูตดิสก์ (Boot Disk) ซึ่งสามารถทำได้โดยขั้นตอนดังนี้

ขั้นตอนการทำแผ่นบูตดิสก์โดยใช้ระบบปฏิบัติการวินโดว์ 95 , 98 หรือ ME

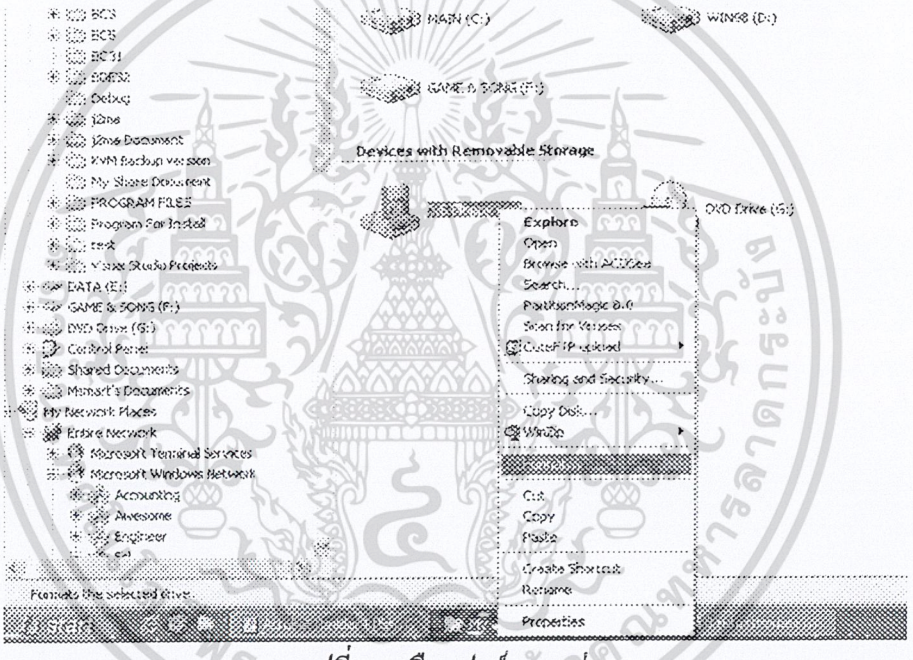
- ใส่แผ่นดิสก์ลงในเครื่อง
- ไปที่ปุ่ม Start แล้วเข้าไปที่ Control Panel
- เลือกเข้าโปรแกรมที่ Add or Remove Programs
- เลือกหัวข้อ Create Startup Disk
- คลิ๊กไปที่ OK

ขั้นตอนการทำแผ่นบูตดิสก์โดยใช้ระบบปฏิบัติการวินโดว์ 2000 หรือ XP

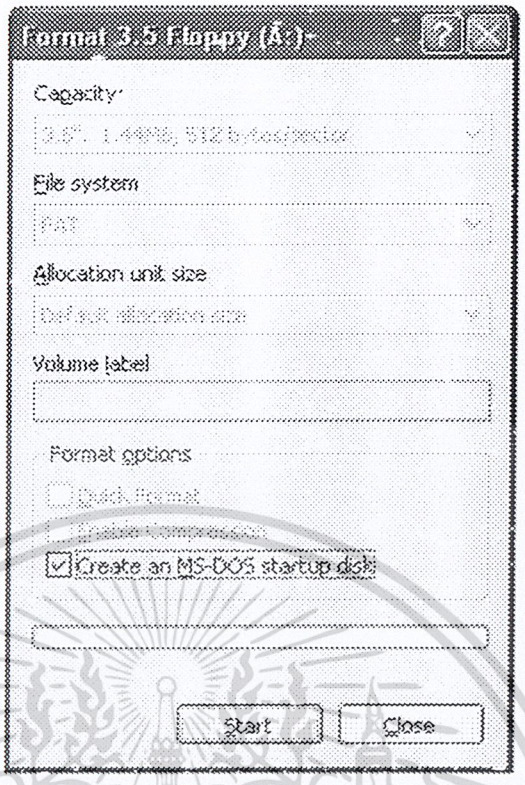
- ใส่แผ่นดิสก์ลงในเครื่อง
- ไปที่ปุ่ม Start แล้วเข้าไปที่โปรแกรม Windows Explores
- เลือกไปที่ My Computer
- คลิ๊กขวาไปที่ไดรฟ์เอ (Drive A)
- เลือกหัวข้อ Format ..
- ที่หัวข้อ Format Option เลือก Create an MS-DOS startup disk
- เลือกทำงานไปที่ Start



รูปที่ 7-1 การทำแผ่นบูตดิสก์โดยใช้ระบบปฏิบัติการวินโดว์ 2000 หรือ XP



รูปที่ 7-2 เลือกฟอร์แมตแผ่น



รูปที่ 7-3 เลือกสร้างแผ่นบูตดิสก์

จากนั้นทำการปิดแล้วเปิดเครื่องขึ้นมาใหม่อีกครั้ง โดยใส่แผ่นดิสก์ลงไปด้วย เครื่องจะทำการเปิดเครื่องโดยเป็นการทำงานระบบดอส จากนั้นทดลองใช้งานเวอร์ชวลแมชีนขนาดเล็กเพื่อทดสอบดูได้

การทดลองใช้งานระบบปฏิบัติการดอสโดยใช้แผ่นบูตดิสก์นั้น เป็นวิธีที่ง่ายกว่าการลงระบบปฏิบัติการดอสลงในเครื่องเอง แต่วิธีนี้จะอาจจะเกิดปัญหาในการทำงานได้ เพราะการใช้แผ่นบูตดิสก์นั้น เป็นเพียงแค่ระบบจำลองของดอสเท่านั้น อาจจะไม่สามารถทำงานได้เมื่อนำไปลงบนบอร์ดคอม 86 ก็ได้

7.2 การใช้งานผ่านอุปกรณ์บอร์ดคอม 86

สิ่งที่ได้มากับชุดพัฒนาบอร์ดคอม 86 มีดังนี้

- อแดปเตอร์สำหรับบอร์ดคอม86 1 อัน
- สาย Serial ชนิด null modem 1 เส้น
- สาย USB ชนิด A-B 1 เส้น
- ซีดีรอมโปรแกรมและคู่มือการใช้งาน 1 แผ่น

ความต้องการของระบบเบื้องต้น

- เครื่องคอมพิวเตอร์สำหรับใช้คอมไพล์และตรวจสอบการทำงานของโปรแกรม 1 เครื่องโดยจะต้องมีพอร์ตอนุกรมสำหรับใช้งานได้อย่างน้อย 1 พอร์ต และมีการ์ดอีเทอร์เน็ต อย่างน้อย 1 การ์ด หากต้องการพัฒนาโปรแกรมที่ใช้คุณสมบัติทางด้านเน็ตเวิร์ค
- โปรแกรมเทอร์มินัล เช่น Hyperterminal เป็นต้น

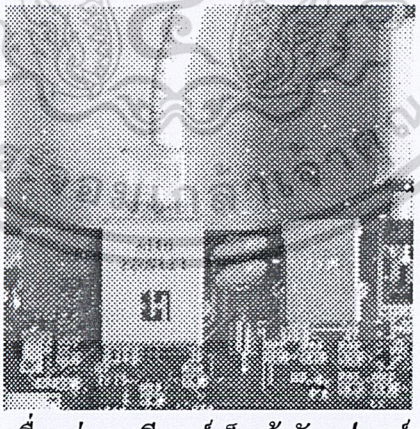
- สาย serial
- สายแลนชนิดแบบสายตรง (straight-through twisted pair cable) และ 10Base-T hub หรือ สายแลนชนิดแบบครอส (twisted pair cross cable)

การติดตั้งการใช้งานบอร์ดคอม 86 มีดังนี้

1. นำอุปกรณ์บอร์ดคอม86 ออกมาจากซองที่ใช้บรรจุเพื่อการขนส่งและตรวจสอบสภาพของบอร์ดใน เบื้องต้นว่าเกิดการชำรุดเสียหายจากการขนส่งหรือไม่
2. เชื่อมต่อสายของ Serial Cable ที่ให้มากับพอร์ตอนุกรมของคอมพิวเตอร์ที่ต้องการใช้งานและเชื่อมต่อสายอีกด้านเข้ากับอุปกรณ์บอร์ดคอม86

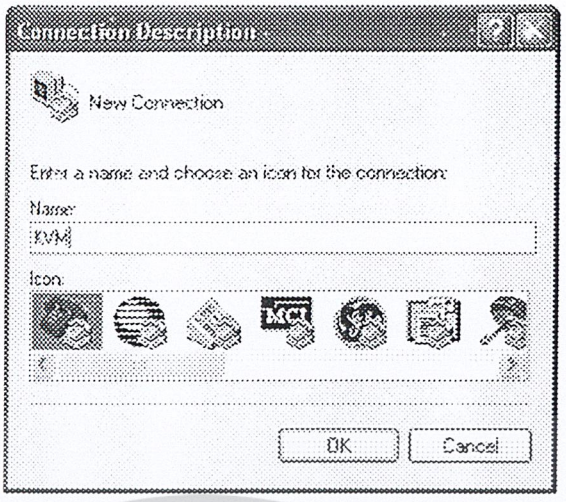
รูปที่ 7-4 การเชื่อมต่อสายซีเรียลเข้ากับอุปกรณ์บอร์ดคอม86

3. เชื่อมต่อสาย Ethernet เข้าอุปกรณ์บอร์ดคอม86 และเครื่องคอมพิวเตอร์หรือเน็ตเวิร์คฮับ



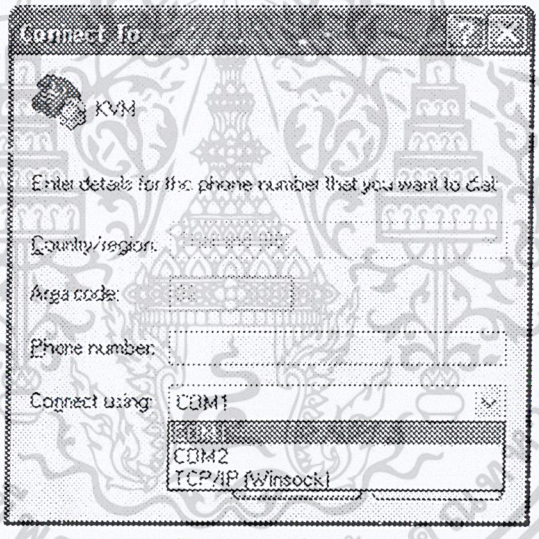
รูปที่ 7-5 การเชื่อมต่อสายอีเทอร์เน็ตเข้ากับอุปกรณ์บอร์ดคอม86

4. เปิดโปรแกรม terminal ที่ใช้งานขึ้นมาและทำการสร้างการเชื่อมต่อใหม่



รูปที่ 7-6 การสร้างการเชื่อมต่อเข้ากับอุปกรณ์บอร์ดคอม86

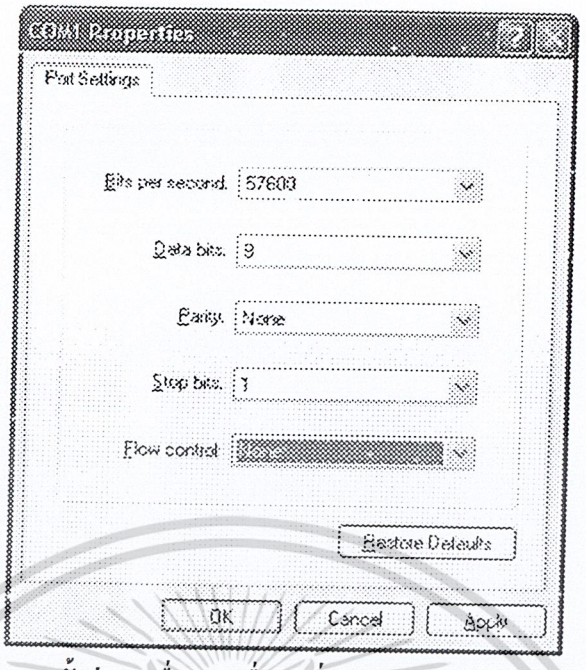
5. จากนั้นเลือกพอร์ตที่ต้องการใช้เชื่อมต่อกับอุปกรณ์บอร์ดคอม86 โดยเลือกที่หัวข้อ Connect using



รูปที่ 7-7 การเลือกพอร์ตที่จะทำการเชื่อมต่อกับอุปกรณ์บอร์ดคอม86

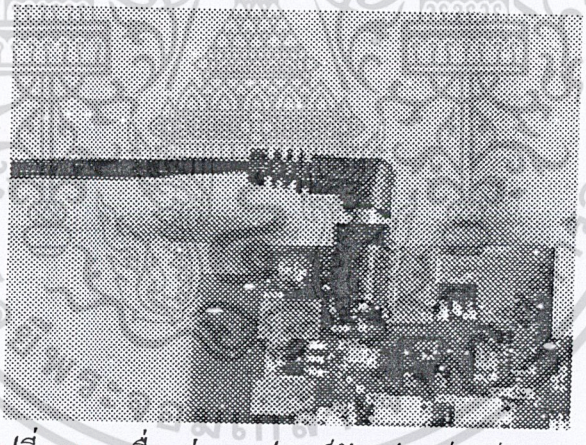
6. จากนั้นทำการตั้งค่าไนการเชื่อมต่อ โดยควรจะตั้งค่าต่างๆดังนี้

- กำหนดอัตราการรับส่งข้อมูลอยู่ที่ 57600 บิตต่อวินาที
- ข้อมูลเป็นแบบ 8 บิต
- ไม่มีพาริตี้
- Stop บิตเท่ากับ 1
- ยกเลิกการทำโฟล์วคอนโทรล (Flow Control) ทั้งทางด้านฮาร์ดแวร์และซอฟต์แวร์



รูปที่ 7-8 การตั้งค่าการสื่อสารเพื่อใช้เชื่อมต่อกับอุปกรณ์บอร์ดคอม86

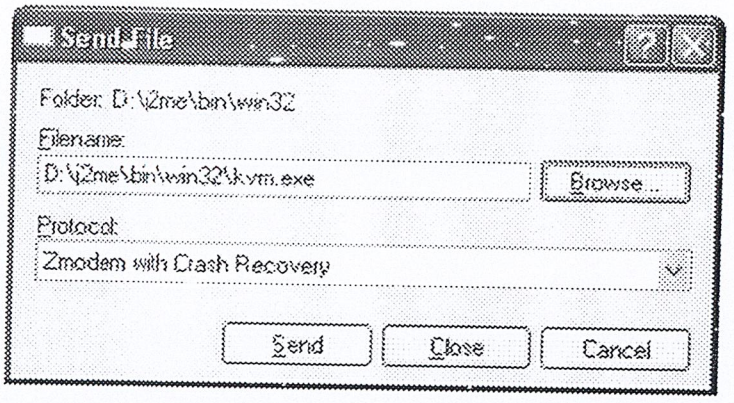
7. ต่อปลั๊กของอแดปเตอร์เข้ากับอุปกรณ์บอร์ดคอม86 และต่ออแดปเตอร์เข้ากับแหล่งจ่ายไฟ



รูปที่ 7-9 การเชื่อมต่ออแดปเตอร์กับอุปกรณ์บอร์ดคอม86

8. กดสวิตซ์รีเซตจะเห็นโปรแกรมบนบอร์ดเริ่มทำงานโดยการแสดงผลผ่านโปรแกรมทอร์มินัลที่ใช้งาน

เมื่อสามารถทำการติดต่อกับบอร์ดคอม 86 ได้แล้ว การนำโปรแกรมเวอร์ชวลแมชีนขนาดเล็กที่เราได้พัฒนาส่งไปที่บอร์ดและเริ่มทดลองใช้งาน การส่งข้อมูลไปที่บอร์ดคอม 86 นั้น สามารถทำได้ด้วยการใช้โปรแกรมทอร์มินัล เลือกไปที่หัวข้อ Tranfer และเลือก Send File จากนั้นจะมีหน้าต่างให้เลือกไฟล์ที่จะทำการส่ง ให้เราเลือกไปที่ไฟล์ที่ต้องการ และกดส่งข้อมูล



รูปที่ 7-10 การส่งไฟล์เข้าไปที่อุปกรณ์บอร์ดคอม86 ผ่านโปรแกรมทอร์มินัล

ส่วนการใช้งานโปรแกรมนั้น ก็ทำได้เช่นเดียวกับที่ใช้งานบนคอมมานพรอมต์ที่ได้ทดลองมาก่อนหน้านี้

7.3 ผลการทดลองทำงาน

หลังจากที่ได้นำโปรแกรมเวอร์ชวลแมชีนขนาดเล็กที่ได้พัฒนามาสำหรับอุปกรณ์บอร์ดคอม86 ลงไปไปที่บอร์ดแล้ว เราก็สามารถทดลองใช้งานโปรแกรมได้โดยผ่านทางโปรแกรมทอร์มินัล โดยผลการทดลองมีดังนี้

7.3.1 การแสดงผลช่วยเหลือต่างๆ

การทดลองใช้คำสั่งแสดงค่าช่วยเหลือ(Help)ต่างๆ ทำได้ดังนี้

- การแสดงค่าช่วยเหลือปกติ โดยพิมพ์คำสั่ง kvm

```
>kvm
ALERT: Must provide class name
Usage: kvm <-options> <classfile>
Options:
  -version
  -classpath <filepath>
  -heapsize <size> <e.g. 65536 or 128k or 1M>
  -debugger
  -suspend
  -nosuspend
  -port <port number>
  -traceallocation
  -tracegc
  -tracegcverbose
  -traceclassloading
  -traceclassloadingverbose
  -traceverifier
  -tracestackmaps
  -tracebytecodes
  -tracemethods
  -tracemethodsverbose
  -tracestackchunks
  -traceframes
  -traceexceptions
  -traceevents
  -tracemonitors
  -tracethreading
  -tracenetworking
  -tracedebugger
  -tracejam
  -traceall <activates all tracing options above>
>
```

รูปที่ 7-11 การแสดงค่าช่วยเหลือ

- การแสดงเวอร์ชันที่กำลังพัฒนาอยู่ โดยใช้คำสั่ง kvm -version

```
>kvm -version
Version: KMITL KVM Version 1.0
```

รูปที่ 7-12 การแสดงเวอร์ชันที่พัฒนาอยู่

7.3.2 การทดลองเรียกคลาสไฟล์ขึ้นมาทำงาน

เนื่องจากการพัฒนาเวอร์ชวลแมชีนขนาดเล็กยังไม่เสร็จสมบูรณ์ดี ทำให้ยังไม่สามารถเรียกคลาสไฟล์ขึ้นมาได้ โดยผลจากการใช้คำสั่ง kvm Hello จะได้ผลการทำงานที่ไม่เสร็จสมบูรณ์ ดังนั้น เพื่อให้ทราบว่าเวอร์ชวลแมชีนขนาดเล็กที่เราพัฒนาขึ้นมาได้ทำงานถึงขั้นไหนแล้ว จึงได้แทรกคำสั่งลงไปในไฟล์ StartJVM.c ดังนี้

```
/* Initialize profiling variables */
InitializeProfiling();
#if INCLUDEDEBUGCODE
  printf("InitializeProfiling Finished\n");
#endif /* INCLUDEDEBUGCODE */

/* Initialize the memory system */
InitializeMemoryManagement();
#if INCLUDEDEBUGCODE
  printf("InitializeMemoryManagement Finished\n");
#endif /* INCLUDEDEBUGCODE */

/* Initialize internal hash tables */
InitializeHashtables();
#if INCLUDEDEBUGCODE
  printf("InitializeHashtables Finished\n");
#endif /* INCLUDEDEBUGCODE */

/* Initialize inline caching structures */
InitializeInlineCaching();
#if INCLUDEDEBUGCODE
```

```
  printf("InitializeInlineCaching Finished\n");
#endif /* INCLUDEDEBUGCODE */

/* Initialize the class loading interface */
InitializeClassLoading();
#if INCLUDEDEBUGCODE
  printf("InitializeClassLoading Finished\n");
#endif /* INCLUDEDEBUGCODE */

/* Load and initialize the Java system classes needed by the VM */
InitializeJavaSystemClasses();
#if INCLUDEDEBUGCODE
  printf("InitializeJavaSystemClasses Finished\n");
#endif /* INCLUDEDEBUGCODE */

/* Initialize the class file verifier */
InitializeVerifier();
#if INCLUDEDEBUGCODE
  printf("InitializeVerifier Finished\n");
#endif /* INCLUDEDEBUGCODE */

/* Initialize the event handling system */
InitializeEvents();
#if INCLUDEDEBUGCODE
  printf("InitializeEvents Finished\n");
#endif /* INCLUDEDEBUGCODE */
```

ซึ่งเมื่อเรียกใช้คำสั่ง kvm Hello จะได้ผลการทำงานดังนี้

```
>kvm Hello
InitializeProfiling Finished
InitializeMemoryManagement Finished
InitializeHashtables Finished
InitializeInlineCaching Finished
InitializeClassLoading Finished
ALERT: Unable to load class java/lang/Object
```

รูปที่ 7-13 การแสดงผลการเรียกคลาสขึ้นมาทำงาน

7.3.3 การทดลองการร้องขอหน่วยความจำ

เวอร์ชวลแมชีนขนาดเล็กที่ได้พัฒนามานั้น สามารถทำงานร้องขอหน่วยความจำได้ โดยค่าการร้องขอหน่วยความจำปกติจะเท่ากับ 65,024 ไบต์ ซึ่งเราสามารถเรียกให้เวอร์ชวลแมชีนขนาดเล็กที่เราพัฒนา แสดงการร้องขอหน่วยความจำได้โดยใช้คำสั่ง kvm –traceallocation <filename> ซึ่งจะได้ผลการทำงานดังนี้

```
>kvm -traceallocation Hello
initializeProfiling Finished
Allocated a dynamic heap of 65024 bytes
Heap bottom: 350a0004, heap top: 350afe04
Allocated 72 bytes, address: 350afdbc, type: 6, free: 64952
InitializeMemoryManagement Finished
InitializeHashtables Finished
InitializeInlineCaching Finished
Allocated 12 bytes, address: 350a01f8, type: 6, free: 64940
Allocated 20 bytes, address: 350a01e4, type: 1, free: 64920
InitializeClassLoading Finished
Allocated 28 bytes, address: 350a01c8, type: 1, free: 64892
Allocated 28 bytes, address: 350a01ac, type: 1, free: 64864
Allocated 24 bytes, address: 350a0194, type: 1, free: 64840
ALERT: Unable to load class java/lang/Object
```

รูปที่ 7-14 แสดงผลการทำงานร้องขอหน่วยความจำ

นอกจากนั้นเรายังสามารถกำหนดขนาดของหน่วยความจำได้เช่นกัน โดยการใช้คำสั่ง –heapsize <heapsize> ซึ่งได้อธิบายไปแล้วในบทที่ 7 โดยลองผลของการใช่งานคำสั่ง kvm –traceallocation –heapsize 50k <file> จะได้ดังนี้

```
>kvm -traceallocation -heapsize 50k Hello
Heapsize = 50
RequestedHeapsize = 51200
InitializeProfiling Finished
Allocated a dynamic heap of 51200 bytes
Heap bottom: 350a0004, heap top: 350ac804
Allocated 72 bytes, address: 350ac7bc, type: 6, free: 51128
InitializeMemoryManagement Finished
InitializeHashtables Finished
InitializeInlineCaching Finished
Allocated 12 bytes, address: 350afff8, type: 6, free: 51116
Allocated 20 bytes, address: 350affe4, type: 1, free: 51096
InitializeClassLoading Finished
Allocated 28 bytes, address: 350affc8, type: 1, free: 51076
Allocated 28 bytes, address: 350affac, type: 1, free: 51048
Allocated 24 bytes, address: 350aff94, type: 1, free: 51020
ALERT: Unable to load class java/lang/Object
```

รูปที่ 7-15 แสดงผลการทำงานร้องขอหน่วยความจำที่กำหนดขนาดไว้

บทที่ 8
บทสรุปและวิจารณ์

การพัฒนาโปรแกรมเวอร์ชวลแมชีนขนาดเล็ก เพื่อให้สามารถทำงานบนบอร์ดคอม86 ที่ใช้ระบบปฏิบัติการดอส จะมีสร้างประโยชน์ต่อผู้ใช้งานอุปกรณ์ต่างๆอย่างมาก การทำงานบนระบบปฏิบัติการดอส ที่ได้รับความนิยมในอุปกรณ์หลายชนิด การเขียนโปรแกรมภาษาจาวาที่ใช้การเขียนโปรแกรมเชิงวัตถุ(Object Oriented Programming) การเขียนโปรแกรมที่เพียงเขียนครั้งเดียวก็สามารถทำงานได้ทุกที่ที่มีเวอร์ชวลแมชีน (Write once run anywhere)

โดยขั้นตอนการพัฒนาเวอร์ชวลแมชีนขนาดเล็กนั้น สามารถสรุปได้ดังนี้

1. ศึกษาเกี่ยวกับเวอร์ชวลแมชีนขนาดเล็กที่ใช้กันอยู่ในปัจจุบัน ศึกษาการใช้งานโปรแกรมเวอร์ชวลแมชีนขนาดเล็ก การใช้งานรูปแบบต่างๆ การนำไปใช้ ประโยชน์ที่ได้จากโปรแกรมและทดลองใช้งานส่วนต่างๆของโปรแกรม
2. ศึกษาการทำงานของอุปกรณ์บอร์ดคอม86 วิธีการควบคุมอุปกรณ์โดยใช้คำสั่งต่างๆ การเขียนโปรแกรมเพื่อควบคุมให้บอร์ดทำงาน ประโยชน์ของอุปกรณ์ คุณสมบัติและข้อจำกัดต่างๆของอุปกรณ์
3. ศึกษาโครงสร้างของโปรแกรมเวอร์ชวลแมชีนขนาดเล็กที่เป็นโค้ดภาษาซี วิธีการคอมไพล์โค้ดเพื่อให้ได้โปรแกรมเวอร์ชวลแมชีนขนาดเล็กที่สามารถทำงานได้ จากนั้นพยายามศึกษาถึงวิธีการนำโปรแกรมมาใช้งานในอุปกรณ์บอร์ดคอม86 ซึ่งใช้ดอสเป็นระบบปฏิบัติการ ซึ่งจะทำได้ยากเนื่องจากโปรแกรมเวอร์ชวลแมชีนขนาดเล็กนั้น ถูกเขียนมาเพื่อให้รองรับการทำงานบนระบบปฏิบัติการ วินโดว์ โซลาลิส ยูนิกซ์ และลีนุกซ์เท่านั้น
4. ศึกษาถึงข้อแตกต่างระหว่างระบบปฏิบัติการวินโดว์ และดอส วิธีการใช้งานต่างๆ เช่น การแสดงผล การร้องขอหน่วยความจำ เป็นต้น จากนั้นทำการพัฒนาโปรแกรมเวอร์ชวลแมชีนขนาดเล็กให้สามารถใช้งานทรัพยากรต่างๆบนดอส และทำงานแสดงผลบนดอสได้ โดยการพัฒนาในส่วนนี้ จะต้องศึกษาถึงไลบรารีต่างๆที่โปรแกรมเวอร์ชวลแมชีน ขนาดเล็กนั้นเรียกใช้งาน ซึ่งบางไลบรารีนั้นจะสามารถทำงานบนดอสได้ แต่บางตัวนั้นจะไม่สามารถทำงานได้ เราจึงต้องพัฒนาทั้งโปรแกรม และแก้ไขตัวไลบรารีบางตัว และทดลองทำงานบนดอสเพื่อตรวจสอบความถูกต้อง จากนั้นนำโปรแกรมที่ได้ไปทดลองใช้งานบนอุปกรณ์บอร์ดคอม86ต่อไป

ข้อแนะนำสำหรับผู้ที่จะพัฒนาโปรแกรมเวอร์ชวลแมชีนขนาดเล็กสำหรับบอร์ดคอม86ต่อไป คือพยายามให้โปรแกรมสามารถใช้งานระบบเน็ตเวิร์คได้ ซึ่งการติดต่อกับส่วนของเน็ตเวิร์คในโค้ดของเวอร์ชวลแมชีนขนาดเล็กนั้น จะสามารถมาพัฒนาให้ใช้กับระบบปฏิบัติการดอสได้ค่อนข้างยาก ทั้งนี้เนื่องจากไฟล์ไลบรารีได้มีการพัฒนาไปค่อนข้างมาก โดยเฉพาะเรื่องของการใช้งานระบบเน็ตเวิร์ค ซึ่งแนวทางการพัฒนาโปรแกรมนั้น อาจจะทำได้ยาก แต่หากลองมาพัฒนาไลบรารีดู อาจจะทำได้ง่ายกว่า

นอกจากนี้ ความสามารถที่เป็นจุดเด่นของภาษาจาวาอีกอย่างหนึ่งคือการทำงานเธรด (Thread) นั้น จำเป็นที่จะต้องให้ระบบปฏิบัติการนั้นๆ ทำงานแบบมัลติทาส์ก (Multitask) ได้ ซึ่งดอสนั้นไม่มีความสามารถนี้ อย่างไรก็ตาม มีโปรแกรมหลายอย่างที่ช่วยให้ดอสนั้นมีความสามารถนี้ ดังนั้นควรศึกษาโปรแกรมเหล่านั้นเพิ่มเติมดู

เนื้อหาทั้งหมดที่ปริญญานิพนธ์เล่มนี้นำเสนอ สามารถนำไปเป็นแนวทางในการพัฒนาโปรแกรมเวอร์ชวลแมชีนขนาดเล็ก เพื่อให้สามารถทำงานบนอุปกรณ์บอร์ดคอม86ได้ ซึ่งการพัฒนาในรูปแบบนี้เรียกอีกอย่างหนึ่งว่าการพอร์ตติงค์ (Porting) นอกจากนั้น ยังสามารถใช้ปริญญานิพนธ์เล่มนี้เป็นแนวทางในการพัฒนาโปรแกรมอื่นๆ เพื่อให้สามารถทำงานบนบอร์ดคอม86ได้อีกด้วย

บรรณานุกรม

[1] William Stallings,Ph.D. : Operating Systems Internals and Design Principles : Prentice-Hall International, Inc. , 2001

[2] Neil Matthew and Richard Stones : Beginning Linux Programming : Wrox Press Ltd, 1999

[3] Jean J. Labrosse : MicroC/OS-II The Real-Time Kernel : Miller Freeman, 1999

[4] Deitel, Harvey M. : Java How To Program : Prentice Hall, 2001

[5] อภิเนตร อุนากูล : Object-Oriented Analysis and Design : โครงการผลิตตำราของคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2543

[6] วีระศักดิ์ ซึงถาวร. 2545. Java Programming Volume. เล่มที่ 1. กรุงเทพฯ : ซีเอ็ดยูเคชั่น

[7] วีระศักดิ์ ซึงถาวร. 2545. Java Programming Volume. เล่มที่ 2. กรุงเทพฯ : ซีเอ็ดยูเคชั่น

[8] ธันวา ศรีประโมง : การเขียนโปรแกรมภาษาซี สำหรับวิศวกรรม : โครงการตำราวิชาการ มหาวิทยาลัยมหานคร, 2538

[9] นุกูล กระจาย : การเขียนโปรแกรมในดอสและวินโดวส์ด้วยบอร์แลนด์C++5.0 : ซีเอ็ดยูเคชั่น, 2540

เว็บไซต์อ้างอิง

http://www.sun.com/	Organize That Build Java and J2ME
http://gear.kku.ac.th/~thana/	Thai Researcher about JAVA
http://www.embedded.com/	Embedded systems programming
http://www.stanford.edu/	Document for network programming
http://www.chipcenter.com/	porting MICROC/OS-II
http://mail.gnu.org/pipermail/make-w32/	How To about Makefile
http://www.gnu.org/software/make/	How To about Makefile
http://www.corej2me.com/	J2ME Community site
http://www.j2me.org	J2ME Community site
http://www.microjava.com/	J2ME Community site
http://www.thaiesf.org/	Com 86 information
http://docsrv.caldera.com:8457/	How to C++ programming
http://www.esl.in.th/	Embeded System Lab,Kmitl

ภาคผนวก

ซอร์สโค้ด VmPort

1. ไฟล์ machine_md.h

```
/*=============================================================================
 * KVM
 *=============================================================================
 * SYSTEM:   KVM
 * FILE:     machin~1.h (for Windows)
 * OVERVIEW: This file is included in every compilation. It contains
 *           definitions that are specific to the Windows port of KVM.
 * AUTHOR:   Frank Yellin
 *           Ioi Lam
 *           Richard Berlin
 * NOTE:     This file overrides many of the default compilation
 *           flags and macros defined in VmCommon/h/main.h.
 *=============================================================================*/

/*=============================================================================
 * Platform definition
 *=============================================================================*/

#define WINDOWS 0

/*=============================================================================
 * Include files
 *=============================================================================*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>
#include <math.h>
#undef CONST
```

```
/*=========================================================================
 * Platform-specific datatype definitions
 *=========================================================================*/
#define BUILD_VERSION "KMITL KVM Version 1.0"
typedef long            Long;
typedef char            Boolean;
typedef long        long64;   /* 64-bit signed integer type */
typedef unsigned long  ulong64;  /* 64-bit unsigned integer type */
typedef long        jlong;    /* Added for KNI */
#ifndef Calendar_md
unsigned long *Calendar_md();
#endif
#define PATH_SEPARATOR   ';'
/*=========================================================================
 * Compilation flags and macros that override values defined in main.h
 *=========================================================================*/
#if ALTERNATIVE_FAST_INTERPRETER
#define ENABLE_JAVA_DEBUGGER 0
#define IPISLOCAL 1
#define SPISLOCAL 1
#define LPISLOCAL 1
#define FPISLOCAL 1
#define CPISLOCAL 1
#endif

/* This is a little-endian target platform */
#define LITTLE_ENDIAN 1

/* Make the VM run a little faster (can afford the extra space) */
#define ENABLEFASTBYTECODES 1

/* Use asynchronous native functions on Windows */
#define ASYNCHRONOUS_NATIVE_FUNCTIONS 0
```

```
/*=========================================================================
 * Platform-specific macros and function prototypes
 *=========================================================================*/

#define InitializeVM()
#define FinalizeVM()
#define freeHeap(x) free(x)
#define RandomNumber_md() rand()
ulong64 sysTimeMillis(void);

/*=========================================================================
 * The following are used in several different places, and its worthwhile
 * to define them just once
 *=========================================================================*/
void* allocateVirtualMemory_md(long size);
void  freeVirtualMemory_md(void *address, long size);
enum { PVM_NoAccess, PVM_ReadOnly, PVM_ReadWrite };
void  protectVirtualMemory_md(void *address, long size, int protection);
int  getpagesize();
/*
 * Things to make general native code build under Windows
 */
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <direct.h>

#ifndef __GNUC__
/*
 * The following is already defined in GCC from the Cygwin32 distribution
 */
#define open    _open
#define close   _close
```

```
#define read    _read
#define write   _write
#define access  _access
#define umask   _umask
#define fstat   _fstat
#define stat    _stat
#define mkdir   _mkdir

#define S_IFMT   _S_IFMT
#define S_IFREG  _S_IFREG
#define S_IFCHR  _S_IFCHR
#define S_IFFIFO _S_IFIFO
#define S_IFDIR  _S_IFDIR
#define S_IFBLK  _S_IFBLK

#define O_BINARY _O_BINARY

#endif /* __GNUC__ */

#ifndef S_ISREG
#define S_ISREG(mode)  ( ((mode) & S_IFMT) == S_IFREG )
#define S_ISCHR(mode)  ( ((mode) & S_IFMT) == S_IFCHR )
#define S_ISFIFO(mode) ( ((mode) & S_IFMT) == S_ISFIFO )
#define S_ISDIR(mode)  ( ((mode) & S_IFMT) == S_IFDIR )
#define S_ISBLK(mode)  ( ((mode) & S_IFMT) == S_IFBLK )
#endif

/*
 * Thread processing routines
 */
void releaseAsyncThread(void);
void processAcyncThread(void);
void Yield_md(void);
```

```
/*=========================================================================
 * FUNCTION:      The stub of GetAndStoreNextKVMEvent
 * TYPE:          event handler
 * OVERVIEW:      Wait for an external event to occur, and store
 *                the event to the KVM event queue.
 *                On a real device, the implementation of this
 *                function should try to conserve battery if
 *                the routine is called with the 'forever'
 *                flag turned on, or with a 'waitUntil' parameter
 *                larger than zero.
 * INTERFACE
 *   parameters:  forever: a boolean flag that tells whether the
 *                KVM has anything to do or not.  When the KVM calls
 *                this routine with the forever flag turned on, the
 *                VM has no threads to run and this function can call
 *                machine-specific battery conservation/hibernation
 *                facilities.  If the forever flag is off, the function
 *                should try to return immediately, or at least after
 *                'waitUntil' milliseconds, since there are other threads
 *                that need to continue running.
 *                waitUntil: the amount of milliseconds this routine
 *                should wait in case the 'forever' flag is off.
 *   returns:     nothing directly, but the event should be stored
 *                in the event queue of the KVM (see events.c and
 *                the porting guide for details).
 *=========================================================================*/

#define GetAndStoreNextKVMEvent(x,y)
#define _NOT_IMPLEMENTED_GetAndStoreNextKVMEvent
```

2. ไฟล์ runtime_md.c

```
/*=========================================================================
 * KVM
 *=========================================================================
 * SYSTEM:    KVM
 * SUBSYSTEM: Machine-specific implementations needed by virtual machine
 * FILE:      runtime_md.c
 * AUTHOR:    Frank Yellin
 *=========================================================================*/

/*=========================================================================
 * Include files
 *=========================================================================*/
#include <global.h>
#include <async.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MEM_COMMIT 0x1000
#define PAGE_READWRITE 0x04
#define MEM_RELEASE 0x8000
#define PAGE_NOACCESS 0x01
#define PAGE_READONLY 0x02
#define PAGE_READWRITE 0x04

/*=========================================================================
 * Functions
 *=========================================================================*/
/*=========================================================================
 * FUNCTION:      alertUser()
 * TYPE:          error handling operation
 * OVERVIEW:      Show an alert dialog to the user and wait for
 *                confirmation before continuing execution.
```

```
 * INTERFACE:
 *   parameters:  message string
 *   returns:     <nothing>
 *=========================================================================*/
void AlertUser(const char* message)
{
   fprintf(stderr, "ALERT: %s\n", message);
}
/*=========================================================================
 * FUNCTION:     allocateHeap()
 * TYPE:         allocates memory
 * OVERVIEW:     Show an alert dialog to the user and wait for
 *               confirmation before continuing execution.
 * INTERFACE:
 *   parameters:  *sizeptr:  INPUT:   Pointer to size of heap to allocate
 *                           OUTPUT:  Pointer to actually size of heap
 *                *realresultptr: Returns pointer to actual pointer than
 *                           was allocated, before any adjustments for
 *                           memory alignment.  This is the value that
 *                           must be passed to "free()"
 *
 *  returns:     pointer to aligned memory.
 *
 *  Note that "sizeptr" reflects the size of the "aligned" memory, and
 *  note the actual size of the memory that has been allocated.
 *=========================================================================*/
cell *allocateHeap(long *sizeptr, void **realresultptr) {
      void *space = malloc(*sizeptr + sizeof(cell) - 1);
      *realresultptr = space;
      return (void *) ((((long)space) + (sizeof(cell) - 1)) & ~(sizeof(cell) - 1));
}
/* Virtual memory allocation and protection operations. */
/* Used for testing the correctness of the garbage collector */
```

```
int getpagesize() {
        return 4096; //Page Size Default}
void*
allocateVirtualMemory_md(long size) {}
void
freeVirtualMemory_md(void *address, long size) {}
void
protectVirtualMemory_md(void *address, long size, int protection)
{
  long o;              /* old protection value */
  int flag;
  switch (protection) {
    case PVM_NoAccess:  flag = PAGE_NOACCESS; break;
    case PVM_ReadOnly:  flag = PAGE_READONLY; break;
    case PVM_ReadWrite: flag = PAGE_READWRITE; break;
    default:        fatalError("Bad argument");
  }
}
/*=========================================================================
 * FUNCTION:     CurrentTime_md
 * TYPE:         Public link routine
 * OVERVIEW:     Get the system time in ms since 1970
 * INTERFACE:
 *   parameters: none
 *   returns:    time
 *=======================================================================*/
ulong64
CurrentTime_md(void) {
  return sysTimeMillis();
}
/*=========================================================================
 * FUNCTION:     InitializeNativeCode
 * TYPE:         initialization
 * OVERVIEW:     Called at start up to perform machine-specific
```

```
 *              initialization.
 * INTERFACE:
 *  parameters: none
 *  returns:    none
 *=============================================================================*/
void InitializeNativeCode() {
   extern void runtime2_md_init(void);
   runtime2_md_init();
}
/*=============================================================================
 * FUNCTION:     nativeFinalization
 * TYPE:         initialization
 * OVERVIEW:     Called at start up to perform machine-specific
 *               finalization.
 * INTERFACE:
 *  parameters: none
 *  returns:    none
 *=============================================================================*/
void FinalizeNativeCode() {}
```

3. ไฟล์ runtime_md2.c

```
#define FALSE 0
#define TRUE 1
/*=========================================================================
 * KVM
 *=========================================================================
 * SYSTEM:   KVM
 * SUBSYSTEM: Machine-specific implementations needed by virtual machine
 * FILE:     runtime_md2.c
 * AUTHOR:   Nik Shaylor
 *
 * NOTE:     These functions have been split out from runtime_md.c
 *           because of conflicts between windows.h and global.h
 *=========================================================================*/
/*=========================================================================
 * Include files
 *=========================================================================*/

#include <machin~1.h>
#include <main.h>

#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MAXCALENDARFLDS 15
#define YEAR 1
#define MONTH 2
#define DAY_OF_MONTH 5
#define HOUR 10
#define MINUTE 12
#define SECOND 13
#define MILLISECOND 14
```

```
static unsigned long date[MAXCALENDARFLDS];

/*=========== ============================================================
 * Functions
 *============================================================================*/
/*============================================================================
 * FUNCTION:     Calendar_md()
 * TYPE:         machine-specific implementation of native function
 * OVERVIEW:     Initializes the calendar fields, which represent the
 *               Calendar related attributes of a date.
 * INTERFACE:
 *   parameters: none
 *   returns:    none
 * AUTHOR:       Tasneem Sayeed
 *============================================================================*/
unsigned long *
Calendar_md(void)
{
  clock_t clk;
  struct tm *tmP = NULL;
  /* initialize the date array */
  memset(&date, 0, MAXCALENDARFLDS);
  /* returns the time */
  time(&clk);
  /* returns pointer to tm struct */
  tmP = localtime(&clk);
  /* initialize the calendar fields */
  date[YEAR] = tmP->tm_year;
  date[MONTH] = tmP->tm_mon;
  date[DAY_OF_MONTH] = tmP->tm_mday;
  date[HOUR] = tmP->tm_hour;
  date[MINUTE] = tmP->tm_min;
  date[SECOND] = tmP->tm_sec;
  /* We cannot accurately determine this value,
```

```
    * so set it to 0 for now
    */
    date[MILLISECOND] = 0;
    return date;
}

/*=========================================================================
 * FUNCTION:      Yield_md()
 * TYPE:          machine-specific implementation of native function
 * OVERVIEW:      Yield the current thread
 * INTERFACE:
 *   parameters:  none
 *   returns:     current time, in centiseconds since startup
 *=======================================================================*/
void Yield_md(void) {}
/*=========================================================================
 * FUNCTION:      runtime2_md_init()
 * TYPE:          Initialization routine
 * OVERVIEW:      Function for initializing the definitions in this file.
 * INTERFACE:
 *   parameters:  none
 *   returns:     none
 *=======================================================================*/

/* We need to make sure runtime2_md_init is only called once when
 * -jam -repeat option is supplied.
 */
static int runtime2_md_inited = FALSE;

void runtime2_md_init(void) {
  if (runtime2_md_inited) {
    return;
  }
#if ASYNCHRONOUS_NATIVE_FUNCTIONS
```

```
    InitializeCriticalSection(&csect);
    beginthreads();
#endif
    runtime2_md_inited = TRUE;
}

#define FT2INT64(ft) \
    ((ulong64)(ft).dwHighDateTime << 32 | (ulong64)(ft).dwLowDateTime)

/*==========================================================================
 * FUNCTION:      sysTimeMillis()
 * TYPE:          time
 * OVERVIEW:      Get system time from Windows.
 * INTERFACE:
 *   parameters:  none
 *   returns:     time in milliseconds
 *==========================================================================*/

unsigned long sysTimeMillis(void) {
        time_t t;
        t = time(NULL);
        return(t*1000);

}
```