User - Mode USB Device Driver บนระบบปฏิบัติการวินโดว์

User - Mode USB Device Driver for Windows

นายวิฑิต เฟื่องพูนทรัพย์

นายอวิรุจน์ เหลืองกิตติก้อง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2548

User - Mode USB Device Driver บนระบบปฏิบัติการวินโดว์

User - Mode USB Device Driver for Windows

โดย

นายวิทิต เฟื่องพูนทรัพย์

นายอวิรุจน์ เหลืองกิตติก้อง

อาจารย์ที่ปรึกษา

อ.เจริญ วงษ์ชุ่มเย็น

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง User - Mode USB Device Driver บนระบบปฏิบัติการวินโดว์

User - Mode USB Device Driver for Windows

ผู้จัดทำ

1. นายวิทิต เฟื่องพูนทรัพย์ เลขประจำตัว 46015373

2. นายอวิรุจน์ เหลืองกิตติก้อง เลขประจำตัว 46015390

อาจารย์ที่ปรึกษา

(อ.เจริญ วงษ์ชุ่มเย็น)

User - Mode USB Device Driver บนระบบปฏิบัติการวินโดว์

นายวิฑิต เฟื่องพูนทรัพย์ 46015373
นายอวิรุจน์ เหลืองกิตติก้อง 46015390
อ.เจริญ วงษ์ชุ่มเย็น อาจารย์ที่ปรึกษา
ปีการศึกษา 2548

บทคัดย่อ

device driver มีการทำงานอยู่ด้วยกัน 2 ส่วน คือ user-mode driver และ kernel-mode driver โดยที่ user-mode driver จะทำหน้าที่ติดต่อกับโปรแกรมสำเร็จรูป และคอยตอบสนองการทำงานต่อโปรแกรมสำเร็จรูป ส่วน kernel-mode driver จะทำหน้าที่ติดต่อกับอุปกรณ์ฮาร์ดแวร์ โดยที่ทั้ง 2 ส่วนจะคุยกันโดยใช้ IRP

ฟังชันที่สำคัญที่ใช้ในการเขียน device driver ที่ใช้ใน user-mode คือ CreateFile(), DeviceIoControl() และ CloseHandle() ฟังก์ชัน CreateFile() เป็นฟังก์ชันที่ใช้ในการสร้างไฟล์ Handle ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อ ฟังก์ชัน DeviceIoControl() เป็นฟังก์ชันที่ใช้ในการติดต่อกับ kernel-mode driver โดยการส่ง IOControl Code ไปให้กับ kernel-mode driver เป็นการส่งข้อมูลไปให้กับ kernel-mode driver เพื่อใช้อ่านข้อมูลมาจากอุปกรณ์ฮาร์ดแวร์หรือเขียนข้อมูลไปยังอุปกรณ์ฮาร์ดแวร์ ฟังก์ชัน CloseHandle() เป็นการปิดไฟล์ Handle ที่ได้สร้างขึ้นมาจากฟังก์ชัน CreateFlie()

User - Mode USB Device Driver for Windows

Mr.Withit Fuengpoonsub	46015373
Mr.Awirut Luangkittikong	46015390
Mr.Charoen Vongchumyen	Advisor
Academic Year 2005	

ABSTRACT

Device Driver have working 2 part. It is user-mode driver and kernel-mode driver. User-mode driver is communication with application software and response working with application software. Kernel-mode driver is communication with hardware. Both user-mode driver and kernel-mode driver will communication by IRP

Function important to program device driver into user-mode driver is CreatFile(). DeviceIoControl() and CloseFile(). Function CreateFile() is function create file handle of hardware. Function DeviceIoControl() is communication with kernel-mode driver. It is send data to kernel-mode driver for read data from hardware or write data to hardware. Function CloseHandle() is close file handle which it create by CreateFile()

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จได้อย่างดี ด้วยคำแนะนำ และคำปรึกษาจาก อ.เจริญ วงษ์ชุ่มเย็น ซึ่งเป็นอาจารย์ผู้ควบคุมปริญญานิพนธ์ ข้าพเจ้ารู้สึกทราบซึ้งในความอนุเคราะห์จากท่านอาจารย์ และขอขอบพระคุณเป็นอย่างสูง

ขอกราบพระคุณคณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุก ๆ ท่านที่ได้ประสิทธิ์ประสาทวิชาให้กับข้าพเจ้า

ขอขอบคุณบริษัท Design Gateway Company Limited,พี่ธนพร แสงไพฑูรย์ และพี่จีระพงษ์ พุทธันบุตรที่ได้สนับสนุนเครื่องมือ ตลอดจนข้อมูล และหนังสือต่างๆ ที่ใช้ในการทำโครงงานนี้

ขอขอบคุณเพื่อนๆ พี่ๆ น้องๆ ในภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยี พระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุกคนที่ให้คำแนะนำต่างๆ และคอยให้กำลังใจเสมอมา

สุดท้ายนี้ข้าพเจ้าขอกราบขอบพระคุณ บิดา มารดา และครอบครัวของข้าพเจ้าที่เป็นกำลังใจและให้การสนับสนุนในทุกๆเรื่อง ทำให้ข้าพเจ้าสามารถทำปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงด้วยดี

คุณค่าและประโยชน์อันพึงมาจากปริญญานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบแด่ผู้มีพระคุณทุกท่าน

นายวิฑิต เพื่องพูนทรัพย์
นายอวิรุจน์ เหลืองกิตติก้อง

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของโครงงาน

ปัจจุบันคอมพิวเตอร์ได้กลายเป็นส่วนหนึ่งของชีวิตประจำวันของผู้คนจำนวนมากซึ่งต้องยอมรับว่าในปัจจุบันคอมพิวเตอร์มีอิทธิพลต่อชีวิตประจำวันมากไม่ว่าจะเป็นในด้านการค้าขาย การติดต่อสื่อสาร หรือว่าจะเป็นการโทรคมนาคม ต่างก็ต้องใช้คอมพิวเตอร์เข้าไปช่วยอำนวยความสะดวกด้วยกันแทบทั้งสิ้น ซึ่งส่งผลให้คอมพิวเตอร์เข้ามามีบทบาทสำคัญในปัจจุบัน ซึ่งจะเห็นได้ว่าในปัจจุบันแทบทุกบ้าน จะมีคอมพิวเตอร์อย่างน้อยหนึ่งเครื่องอยู่ในบ้าน

ซึ่งคอมพิวเตอร์นั้นภายในก็จะมีอุปกรณ์มากมายที่ซับซ้อนอยู่ภายใน และอุปกรณ์เหล่านั้นก็จะต้องมี ระบบปฏิบัติการหรือที่เรียกว่า Operating System (OS) ไว้คอยควบคุมการทำงานของอุปกรณ์เหล่านั้น ซึ่งระบบปฏิบัติการหรือ Operating System นั้น บางคนอาจจะรู้จักแค่ Windows หรือ Linux แต่อันที่จริงแล้วระบบปฏิบัติการในปัจจุบันนั้นมีอยู่เป็นจำนวนมาก ไม่ว่าจะเป็น OS2 , Solaris เป็นต้น โดยที่ผู้คนส่วนใหญ่ในประเทศไทย จะรู้จักระบบปฏิบัติการกันอยู่2 ระบบ คือ Windows และ Linux ซึ่งทั้งสองระบบนี้เป็นที่ได้รับความนิยม โดยที่ระบบปฏิบัติเหล่านี้นั้น ต่างก็ไม่สามารถที่จะติดต่อกับ Hardware ได้โดยตรงจึงต้องมีการสร้างโมดูล ขึ้นมาเพื่อให้ระบบปฏิบัติการนั้นสามารถติดต่อกับ Hardware ได้ ซึ่งโมดูลนั้นจะถูกเรียกว่า Device Driver โดยจะทำหน้าที่เป็นสื่อการในการติดต่อระหว่าง Hardware กับ Software ให้สามารถทำงานร่วมกันได้อย่างมีประสิทธิภาพ

โดยจะเห็นว่าอุปกรณ์แต่ละตัวที่ทำการติดต่อสื่อสารอยู่กับระบบปฏิบัติการนั้น ต่างก็ต้องมี Device Driver เป็นของตัวเอง เพื่อจะได้ติดต่อสื่อสารกันได้ถูกต้อง ดังนั้นเราจึงมีความสนใจที่จะศึกษาเกี่ยวกับ Device Driver เพื่อนำไปประยุกต์ใช้ต่อในอนาคต ไม่ว่าจะเป็นในด้านการศึกษาต่อ หรือทางด้านธุรกิจ

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาเกี่ยวกับการทำงานระหว่าง Operating System กับ Device Driver
2. เพื่อศึกษาการติดต่อและการทำงานภายใน Device Driver
3. เพื่อศึกษาการเขียน User-Mode Driver ในการติดต่อกับ Operating System
4. เพื่อเขียน User-Mode Driver ไปติดต่อกับ Kernel-Mode Driver เพื่อติดต่อ Hardware

1.3 ขอบเขตของโครงงาน

1. ศึกษาการทำงานภายในของ Device Driver
2. ศึกษาการติดต่อและการทำงานระหว่าง Operating System กับ Device Driver
3. ศึกษาการเขียน User-Mode Driver ติดต่อกับ Kernel-Mode Driver เพื่อติดต่อกับ Hardware
4. เขียน User-Mode Driver ติดต่อกับ Kernel-Mode Driver เพื่อติดต่อกับ Hardware
5. เขียน User-Mode Driver ไปติดต่อกับ register

1.4 วิธีการดำเนินการ

1. ศึกษาเกี่ยวกับการทำงานระหว่าง Operating System กับ Device Driver
2. ศึกษาการติดต่อและการทำงานภายใน Device Driver
3. ศึกษาการเขียน User-Mode Driver ในการติดต่อกับ Operating System
4. จัดหาวัสดุอุปกรณ์ที่จำเป็นต้องใช้ในการพัฒนา
5. วิเคราะห์ และออกแบบระบบ
6. เขียน User-Mode Driver ไปติดต่อกับ Kernel-Mode Driver เพื่อติดต่อกับ Hardware
7. แก้ไขส่วนที่ผิดพลาดเพื่อให้สามารถทำงานได้สมบูรณ์มากที่สุด

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. เข้าใจโครงสร้างการทำงานระหว่าง Operating System กับ Device Driver
2. เข้าใจหลักการทำงานภายใน Device Driver
3. เข้าใจหลักการเขียน User-Mode Driver
4. สามารถเขียน User-Mode Driver เพื่อไปติดต่อกับ Kernel-Mode Driver เพื่อติดต่อกับ Hardware ได้

1.6 ส่วนประกอบของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้ได้แบ่งเนื้อหาออกเป็น 5 บทด้วยกันคือ

บทที่ 1 กล่าวถึงความสำคัญและที่มาของโครงงาน วัตถุประสงค์ของโครงงาน ขอบเขตของโครงงาน วิธีการดำเนินการ ประโยชน์ที่คาดว่าจะได้รับ และส่วนประกอบของปริญญานิพนธ์

บทที่ 2 กล่าวถึงทฤษฎีพื้นฐานที่ใช้ในโครงงาน ซึ่งประกอบด้วยทฤษฎีอะไรบ้าง บรรยายทฤษฎีทั้งหมดโดยละเอียด

บทที่ 3 กล่าวถึงชิ้นงานของโครงงานนี้ ส่วนที่ได้พัฒนาขึ้น การทำงานของระบบหรือชิ้นงานบรรยายโดยละเอียด

บทที่ 4 กล่าวถึงการทดลองและผลการทดลอง

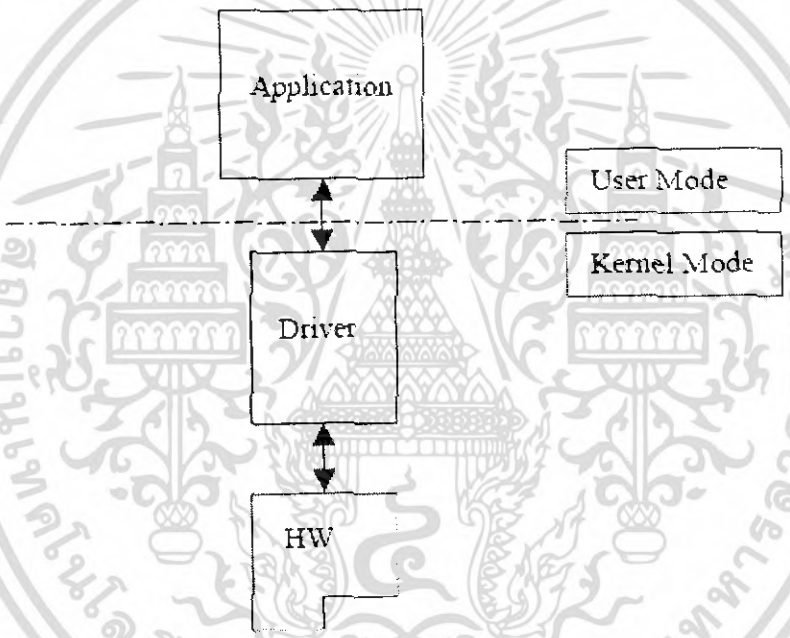
บทที่ 5 เป็นบทวิจารณ์และสรุป ซึ่งกล่าวถึงบทสรุปของโครงงาน วิจารณ์สิ่งที่ได้รับจากโครงงาน และข้อเสนอแนะสำหรับเป็นแนวทางในการพัฒนาต่อ

บทที่ 2
ทฤษฎีพื้นฐาน

2.1 ทฤษฎีในส่วนของ device driver

2.1.1Device Driver Overview

ปัจจุบันคอมพิวเตอร์มีระบบปฏิบัติการหรือที่เรียกกันว่า Operating System (OS)อยู่มากมายไม่ว่าจะเป็น Windows , Linux ,OS2 หรือ Solaris เป็นต้น ซึ่งในระบบปฏิบัติการต่างก็มีระบบการจัดการภายในระบบ ซึ่งการที่OS นั้นจะทำการติดต่อกับhardwareนั้น จะไม่สามารถทำการติดต่อสื่อสารได้โดยตรง ดังนั้น จึงจะต้องมีการติดต่อผ่านกระบวนการที่เรียกว่า"Device Driver" ซึ่งจะเป็นสื่อกลางที่คอยติดต่อสื่อสารระหว่าง application กับ hardware



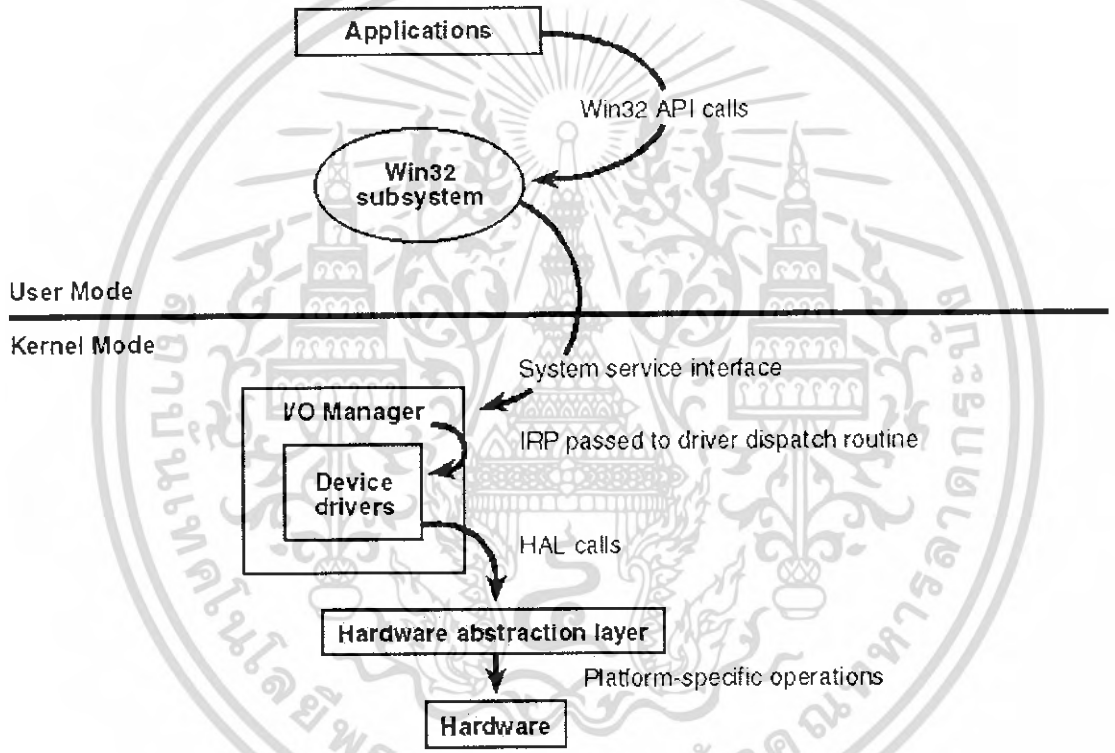
รูปที่ 2.1 แสดงความสัมพันธ์ของ Device Driver กับ ระบบ

จากรูปที่ 2.1 จะเห็นได้ว่ารูปถูกแบ่งออกเป็น2ส่วนคือ User Mode และ Kernel Mode ซึ่ง 2 ส่วนนี้จะมีหน้าที่การทำงานที่ต่างกัน ซึ่ง Application นั้นจะอยู่ในส่วนของ User Mode ซึ่งในส่วนนี้จะไม่สามารถที่จะติดต่อ เช่น อ่านหรือเขียนไฟล์ได้โดยตรง ทุกอย่างที่จะทำเกี่ยวกับhardware นั้นจะต้องทำผ่าน Device Driver ซึ่งอยู่ในส่วนของ Kernel Mode ซึ่งทั้งสองส่วนนี้จะมีลำดับความสำคัญที่ไม่เท่ากัน คือในส่วนของ User Mode นั้นจะไม่สามารถที่จะติดต่อกับ hardware ได้โดยตรง ซึ่งต่างจากในส่วนของ Kernel Mode ที่สามารถที่จะทำการติดต่อกับ hardware ได้โดยตรง

อย่างไรก็ตาม ในแต่ละระบบก็จะมี Layer ลำดับการทำงานที่แตกต่างกันออกไปแล้วแต่โครงสร้างของระบบนั้นที่ถูกออกแบบมาว่าจะมีการทำงานอย่างไร อย่างเช่นที่เราจะยกตัวอย่างเอามา 2 ตัวอย่าง เพื่อให้เห็นถึงหลักการทำงานของระบบปฏิบัติการ ตัวอย่างที่เราจะนำมาให้ในการเสนอก็คือ Windows 98 และ Windows 2000

2.1.2.Windows 2000

Windows 2000 Operating System Software ใดที่จะทำการ excute นั้น อย่างเช่น Application ที่ต้องการจะอ่านข้อมูลบางตัวจากอุปกรณ์ จะต้องไปเรียก Application Programming Interface (API) ไปเรียกที่ Subsystem Module เพื่อใช้ System service interface เพื่อที่จะไปติดต่อกับ I/O Manager



รูปที่2.2 แสดง Architecture ของ Windows 2000

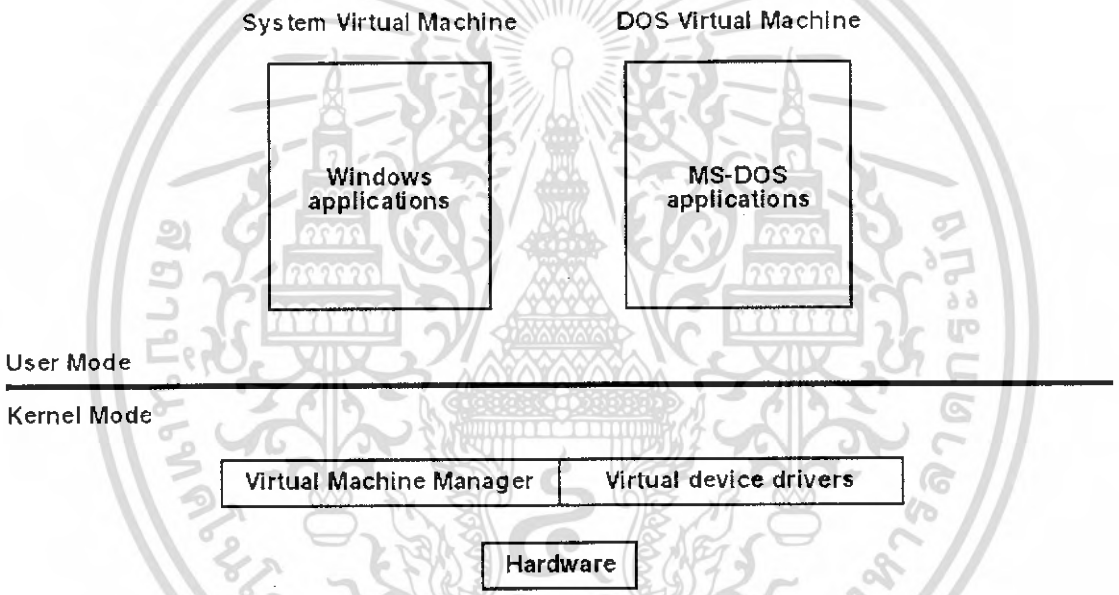
User Mode นั้นไม่สามารถ access ได้ด้วยตัวมันเอง พวกเขาจึงได้สร้าง data structure ที่เรียกว่า I/O request packet (IRP) เป็นตัวสื่อสารกับ Device Driver ซึ่ง Device Driver นั้นจะไปติดต่อผ่านทาง Hardware Abstraction Layer (HAL)เพื่อติดต่อกับ hardware อีกทีหนึ่ง ซึ่ง HALนั้นจะเป็นตัวที่จะไปติดต่อกับ hardware จริงๆ ดังนั้นเราจะเห็นได้ว่าการทำงานของ Windows2000 นั้น การทำงานหลักๆ

นั้นจะทำงานอยู่ในส่วนของ I/O Manager ซึ่งจะเป็นคนคอยติดต่อสื่อสารกับ Device Driver ให้กับระบบปฏิบัติการ

2.1.3. Windows 98

รูปนี้แสดงให้เห็นถึงแนวคิดของ Windows 98 จากรูปที่3 จะพบว่ามันถูกแบ่งออกเป็น 2 ส่วนคือ ส่วนของ System Virtual Machine และ DOS Virtual Machine โดยที่ Kernel Mode จะเป็นตัวเรียก Virtual Machine Manager (VMM) เพราะว่า งานหลักของมันคือการสร้าง virtual machine ขึ้นมา Windows 98 ไม่สามารถจัดการ I/O ได้เหมือนกับ Windows2000ที่มีตัว I/O Managerคอยดูแลเรื่องของ I/O

ส่วนแตกต่างที่สำคัญของWindows 98 ที่ต่างจาก windows2000 คือ การที่มีการจัดการติดต่อโดยตรงกับ disk หรือ port



รูปที่2.3 แสดงแนวคิดของ Windows98

2.1.4.Basic Structure ของ WDM Driver

ในส่วนของ basic structure ของ WDM Driver เราจะพูดกันถึงใน 3 เรื่องหลักๆที่จำเป็นต้องใช้งาน ก็คือ

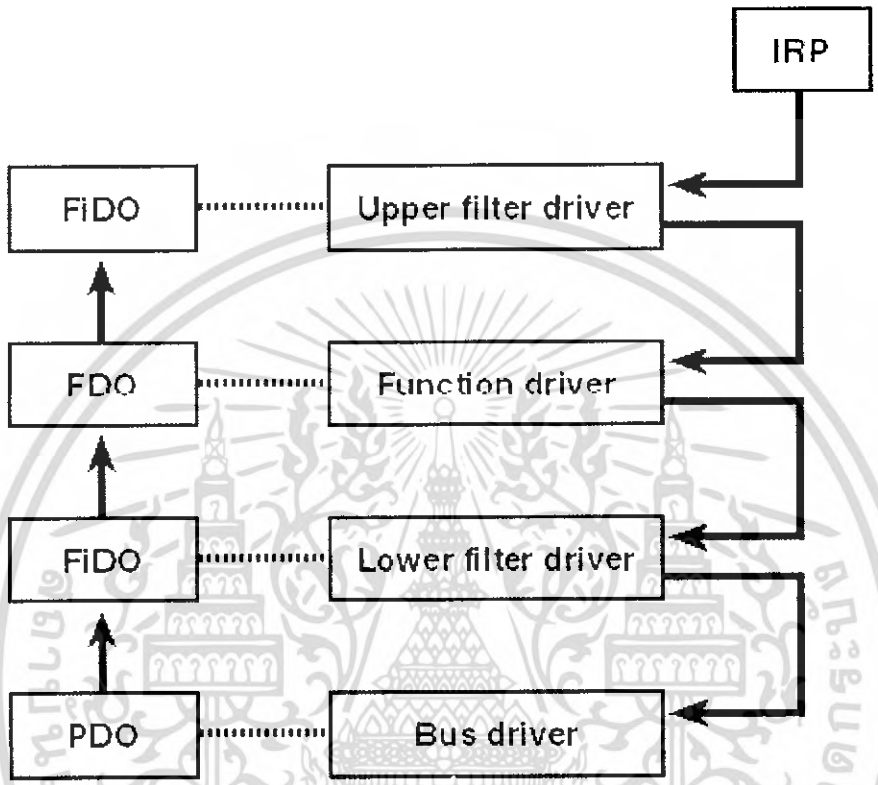
2.1.4.1. Device and Driver Layer

2.1.4.2. DriverEntry Routine

2.1.4.3. AddDevice Routine

ซึ่งทั้ง 3 ส่วนนี้จะเป็นพื้นฐานเพื่อใช้ในการเขียน Device Driver ขึ้นมา ซึ่งเราจะต้องนำไปใช้งานต่อในเรื่องต่อๆไป

2.1.5. Device and Driver Layer



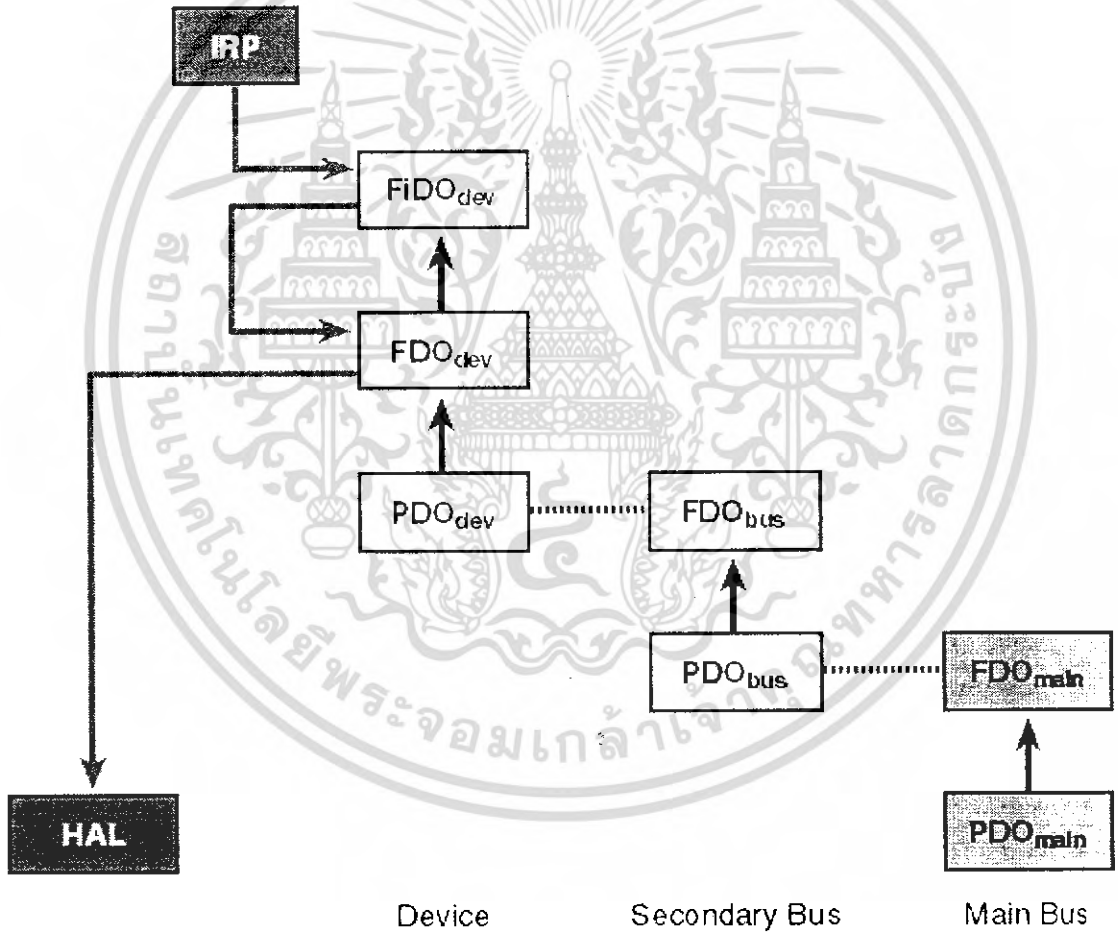
รูปที่2.4 แสดง layer ของ device object และ driver ใน WDM

เป็นรูปที่ 2.4 layerของWindows Driver Model of Driver Stackของdevice object เป็น data structureซึ่งSystem เป็นคนสร้างขึ้นมาเพื่อช่วยให้ softwareที่จะไปจัดการกับhardware ที่ชั้นล่างสุดของ device object ใน stack ที่เรียกว่า physical device object หรือ PDO ต่อมาเราจะมาดูในส่วนตรงกลางของdevice object stack ซึ่งเป็นส่วนของobjectที่เรียกว่า function device object หรือFDO ซึ่งส่วนที่อยู่ด้านบนหรือด้านล่างของFDOจะสามารถเก็บส่วนของfilter device object ซึ่งอยู่เหนือFDOเราจะเรียกว่า upper filter และด้วยเหตุนี้ filter device object ที่อยู่ด้านล่างของFDOจึงถูกเรียกว่า lower filter

โดยที่ Plug and Play (PnP) Manager ของระบบปฏิบัติการจะเป็นคนสร้างstack ของdevice object การที่จะสร้างPDOนั้น PnP Manager จะต้องพิจารณาที่ Registry database และfunction driver ที่อยู่ตรงกลางของรูป จะมีการติดตั้งโปรแกรมเพื่อตอบสนองกับ Registry จำนวนมากที่จะเข้ามา โดยที่ Registry

Entry จะเป็นตัวที่คอยกำหนดว่า Driverตัวไหนที่จะไปแสดงอยู่ในstack ดังนั้นก่อนที่PnP Manager จะไปโหลด Lowest filter driver และเรียก ฟังก์ชั่น AddDevice มันก็จะสร้าง FiDO และสร้างเส้นทางเพื่อไปติดต่อระหว่าง FiDOและDriver AddDevice ที่ต่อจาก PDO ไปยัง FiDO

วัตถุประสงค์ของการทำlayerจะปรากฏชัดเจนขึ้นเมื่อคุณพิจารณาการส่งผ่านของ IRP ซึ่งโดยปรกติแล้วIRPจะถูกส่งผ่านไปที่ส่วนบนสุดของ stack แล้วกรองผ่านลงมายังstackอื่นต่อไป ซึ่งdriver จะเป็นตัวที่ตัดสินใจว่าจะทำอะไรกับIRPบ้าง ซึ่งบางครั้งอาจจะไม่จำเป็นต้องทำไปถึงชั้นล่างสุดของ stack โดยที่IRPอาจจะทำลงมาแค่ชั้นเดียวแล้วเสร็จเลยก็ได้ โดยสังเกตุได้จากรูปที่ 2.2 ซึ่งจะแสดงการทำงานของ device object ที่มีการส่งผ่านค่า IRP ไปยัง FiDO เพื่อทำการกรองการทำงาน ซึ่งตัวอย่างนี้จะเป็นการที่อุปกรณ์ตัวหนึ่งต้องการที่จะติดต่อกับส่วนของ HAL จะพบว่าเมื่อมันส่ง IRP ไปที่ Upper filter แล้วทำการส่งต่อไปที่FDO เพื่อทำการติดต่อกับ HAL โดยที่มันยังไม่ได้ติดต่อกับPDO



รูปที่2.5 แสดงตัวอย่างการทำงานของ device object

2.1.6. DriverEntry Routine

ในส่วนก่อนหน้านี้ เราได้กล่าวไปว่า PnP Manager จะโหลดdriver ที่จำเป็นสำหรับhardware และเรียก AddDevice function ขึ้นมา ให้driverนั้นสามารถใช้กับhardwareได้และการทำglobal initial จะทำเพียงครั้งเดียวเท่านั้น เมื่อตอนที่โหลดขึ้นมาครั้งแรก ซึ่งglobal initial จะรับหน้าที่ในส่วนของ DriverEntry routine

DriverEntry เป็นชื่อที่ถือเอาตามแบบธรรมดาที่ตั้งมาให้เป็นจุดที่เข้าไปสู่ Kernel Mode driver I O Manager เรียก routineดังนี้

```
extern "C" NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject,
  IN PUNICODE_STRING RegistryPath)
  {
  ...
  }
```

ก่อนที่จะพูดถึงcodeที่อยู่ภายในDriverEntry เราต้องการพูดอ้างอิงถึงสิ่งสำคัญเล็กๆน้อยในเรื่องของ*function prototype* ของตัวมันเอง เราจะใช้คำสั่ง extern "c" เพราะว่ามันเป็นกฎ เราจะใส่โค้ดของในรูปแบบของ C++

*WDM ทำหน้าที่หลักๆในDriverEntry คือการเติมfunction point*ต่างๆในdriver object pointer จะชี้ไปยังที่การทำงานที่พบsubroutine ซึ่งประกอบด้วยpointerที่เป็นสมาชิกของDriver Object

- DriverUnload
  ต้อง set pointer นี้ไปที่ cleanup routine สร้าง I/O manager ควรจะเรียก routine นี้เมื่อต้องการ unload ไดรเวอร์ WDM driver ไม่ได้จาก resource ให้ใน DriverEntry ดังนั้น มันจึงไม่จำเป็นที่จะต้อง cleanup ทุกครั้ง
- DriverExtension->AddDevice
  เซ็ต pointer นี้ไปที่ AddDevice function ของคุณ PnP manager จะเรียก AddDevice ของแต่ละฮาร์ดแวร์ที่ตอบสนองคุณ ขณะที่ AddDevice มีความสำคัญในการทำงานของ WDM driver
- DriverStartIo
  ถ้าไดรเวอร์คุณใช้ standard method ของ I/O request Queue คุณควรจะเซ็ทให้ member นี้ของ driver object ชี้ไปที่ StartIo routine ของคุณ
- MajorFunction
  ตัว I/O manager มีการ initialize เวกเตอร์นี้ของฟังก์ชั่น pointer to pointer to dispatch function ซึ่งมีการร้องขอล้มเหลวตลอด คุณควรจะเข้าไปในส่วนของเซนิคของ IRP ไดรเวอร์ของคุณ

```
      extern "C" NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject,
        IN PUNICODE_STRING RegistryPath)
        {
1 →     DriverObject->DriverUnload = DriverUnload;
        DriverObject->DriverExtension->AddDevice = AddDevice;
        DriverObject->DriverStartIo = StartIo;
2 →     DriverObject->MajorFunction[IRP_MJ_PNP] = DispatchPnp;
        DriverObject->MajorFunction[IRP_MJ_POWER] = DispatchPower;
        DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTROL] = DispatchWmi;
3 →     ...
4 →     servkey.Buffer = (PWSTR) ExAllocatePool(PagedPool,
          RegistryPath->Length + sizeof(WCHAR));
        if (!servkey.Buffer)
          return STATUS_INSUFFICIENT_RESOURCES;
        servkey.MaximumLength = RegistryPath->Length + sizeof(WCHAR);
        RtlCopyUnicodeString(&servkey, RegistryPath);
5 →     return STATUS_SUCCESS;
        }
```

1.จะเห็นว่าทั้ง3 statementจะset function pointerให้กับ entry pointer ในdriver โดยเราได้เลือกเอาชื่อฟังก์ชั่นง่ายๆ คือ DriverUnload , AddDeviceและ StartIo

2.WDM driverทุกๆตัว ต้องมีการจัดการ PNP,POWER และ SYSTEM_CONTROL I/O request

3.ในส่วนนี้เราละไว้ในฐานที่เข้าใจ มันคือส่วนที่เราต้องมาเขียนเสริมต่อเอาเอง

4.ถ้าคุณจำเป็นต้องติดต่อกับ service registry key อื่นในDriverของคุณ มันเป็นความคิดที่ดีที่copyเอา RegistryPath ถ้าคุณจะไปทำงานที่WMI(Windows Management Instrumentation)
ซึ่งในตัวอย่างเราสมมุติว่าคุณประกาศ global variable ที่มีชื่อว่า servkey เป็นUNICODE_STRING

5.การreturn STATUS_SUCCESS เป็นวิธีที่คุณแสดงว่ามันทำเสร็จแล้ว ถ้าคุณตรวจพบว่ามีการผิดพลาดมันจะreturn error code จาก NTSTATUS.H หรือจากsetของerror codeซึ่งคุณเป็นคนกำหนดเอง

DriverUnload

วัตถุประสงค์ของWDM driverที่เป็น DriverUnload functionเป็นการclean up หลังจาก global initializationใดๆ ซึ่ง DriverEntry สามารถทำได้

2.1.7.AddDevice Routine

ในส่วนของก่อนหน้านี้ เราได้แสดงให้คุณเห็นถึงวิธีการinitialize ที่WDM driver เมื่อครั้งที่เริ่มทำการโหลดครั้งแรก โดยทั่วไปนั้นdriverนั้นสามารถเรียกหรือจัดการdeviceมากกว่าหนึ่งdevice ในWDM architecture driverมีฟังก์ชั่นพิเศษชื่อว่า AddDeviceซึ่ง PnP Managerสามารถเรียกใช้แต่ละdeviceได้ ซึ่งfunctionมีprototype ดังนี้

```
NTSTATUS AddDevice(PDRIVER_OBJECT DriverObject, PDEVICE_OBJECT pdo)
  {
  }
```

driver objectนี้เป็นจุดเดียวกับdriver objectที่ถูกinitialในDriverEntry routine ของคุณ ซึ่ง PDO คือ addressของphysical device object ที่อยู่ด้านล่างของstack

พื้นฐานการตอบสนองของAddDeviceในfunction driver เป็นการสร้างdevice objectและlinkมันไปสู่ stack root ใน PDO ซึ่งมีขั้นตอนดังนี้

2.1.7.1. เริ่มโดยการเรียก IoCreateDeviceในการสร้างdevice objectและกรณีของdevice objectที่เป็นส่วนเพิ่มเติมมาของคุณ

2.1.7.2. Applicationรู้ว่ามีRegister กี่ตัวที่อยู่ในdeviceของคุณ ในทางกลับกันเราให้device object nameและสร้างsymbolic link

2.1.7.3. ต่อมาให้initalize deviceที่เพิ่มเติมของคุณและflag memberของdevice object

2.1.7.4. เรียก IoAttachDeviceToDeviceStackวางลงในdevice objectตัวใหม่ไปสู่ stack

ดังนั้นเราจะมาดูในรายละเอียดขั้นตอนของ Creating a Device Object ซึ่งคุณสร้าง device object โดยการเรียก IoCreateDevice ดังตัวอย่าง

```
PDEVICE_OBJECT fdo;
NTSTATUS status = IoCreateDevice(DriverObject,
  sizeof(DEVICE_EXTENSION), NULL,
  FILE_DEVICE_UNKNOWN, FILE_DEVICE_SECURE_OPEN, FALSE, &fdo);
```

ในArgrumentแรกเราจะมาดูกันที่ Driverobject จะเป็นตัวเดียวกันกับAddDevice ซึ่งDriverobject จะสร้างการติดต่อระหว่างdriverของคุณและ device objectตัวใหม่ ดังนั้น มันจะยอมให้I/OManagerส่ง IRPไปให้device

ในส่วนต่อมาargumentนี้เป็นขนาดของdeviceที่ถูกสืบทอดมา ในส่วนนี้เราได้อธิบายอย่างง่ายๆซึ่งในส่วนของI/O Manager ได้กำหนดการเพิ่มmemoryและset DeviceExtension pointer ใน device object ไปที่pointerของมัน

Argrumentที่สามซึ่งเป็น NULLในตัวอย่างสามารถเป็นaddressของ UNICODE_STRING จัดไว้ให้device object ตัดสินใจ

Argrumentที่สี่(FILE_DEVICE_UNKNOW) เป็นส่วนหนึ่งของ device type ซึ่งค่าอะไรก็ตามของคุณกำหนดที่นี่ คุณสามารถoverridden โดยเข้าไปในdeviceที่เป็นhardware key หรือclass key ถ้ามีทั้งคู่ overrideเราจะให้ hardware key มีอภิสิทธิ์ที่มากกว่า

Argumentที่ห้า(0) จัดเตรียมCharacteristic flagไว้ให้ device object ซึ่งflagเหล่านี้มีความสัมพันธ์กับ mass storage device ซึ่ง FILE_AUTOGENERATED_DEVICE_NAMEจะไม่มีเอกสารอ้างอิง จะมีอยู่แต่ในการใช้ภายในเท่านั้น เพราะ เอกสารของ DDK ลืมพูดถึงมัน อย่างใดก็ตาม valueที่คุณกำหนดที่นี้สามารถ overridden โดยเข้าไปในdevice hardware key หรือ class key

Argumentที่หก IoCreateDevice (ในตัวอย่างเป็น FALSE) แสดงให้ว่าdeviceเป็นexclusive ซึ่ง I/O Manager จะยอมให้เข้าไปจัดการได้เพียงตัวเดียวเท่านั้นโดยexclusive device อย่างใดก็ตาม valueที่คุณกำหนดที่นี้สามารถ overridden โดยเข้าไปในdevice hardware key หรือ class key

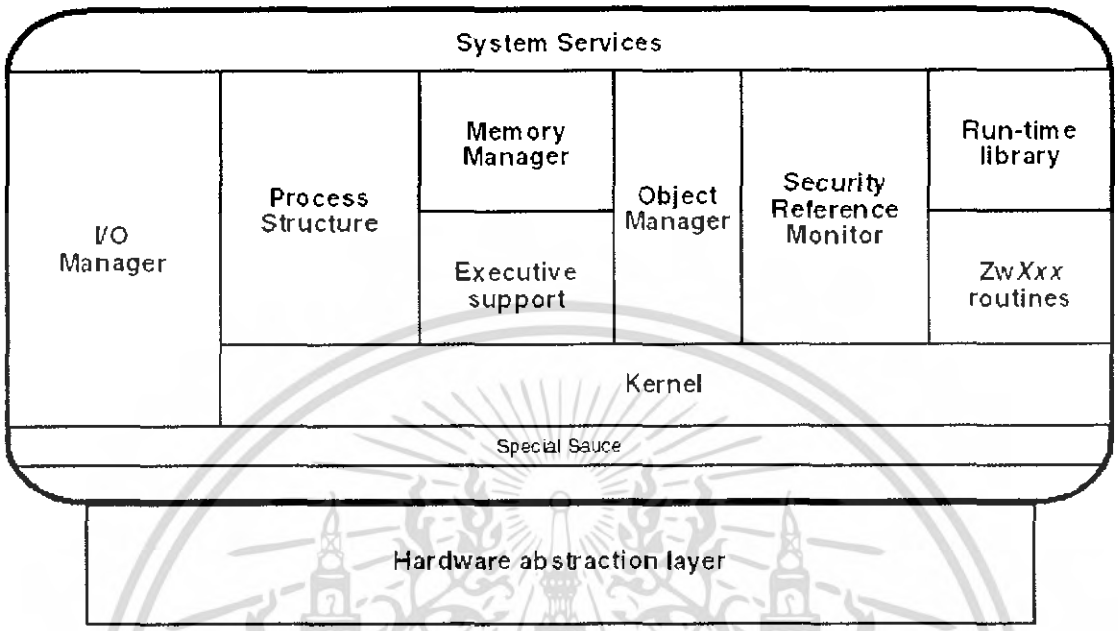
Argumentสุดท้าย (&fdo) เป็นตำแหน่งที่ IoCreateDevice จะเก็บ addressของdevice objectที่มันสร้าง

ถ้า IoCreateDevice ล้มเหลวในบางสาเหตุ มันจะreturn ค่า status code และไม่ทำการเปลี่ยนแปลง PDEVICE_OBJECT แต่ถ้ามัน succeed มันจะ return ค่า successfulของstatus code และset ค่า PDEVICE_OBJECT คุณสามารถกระทำต่อไปโดยการinitialize deviceของคุณที่สืบทอดมาและทำงานอย่างอื่นที่เกี่ยวกับการสร้าง device objectตัวใหม่ คุณควรจะค้นพบerror หลังจากจุดนี้ไปและคุณควรจะแก้ไขdevice object และ return status code

```
NTSTATUS status = IoCreateDevice(...);
if (!NT_SUCCESS(status))
  return status;
...
if (<some other error discovered>)
  {
  IoDeleteDevice(fdo);
  return status;
  }
```

2.2 Basic Program

2.2.1.Basic Programming techniques



รูปที่2.6 แสดง kernel support routine

ในส่วนนี้เราจะมาพูดถึงเรื่องทั่วไปของการเขียน kernel mode ดังรูปที่2.6 แสดงบางส่วนของ Microsoft Windows NT operatingsystem ซึ่งในแต่ละส่วนเป็นfunction การบริการซึ่งมีชื่อนำหน้าอยู่2 ตัว ดังนี้

- **I/O Manager (Io)** เป็นตัวที่เก็บ function จำนวนมากซึ่งdriver เป็นคนเรียกใช้
- **Process Structure module(Ps)** เป็นตัวสร้างและจัดการ kernel-mode thread ซึ่งโดยปกติแล้วWDM driver สามารถจะใช้ thread ได้อย่างอิสระ
- **Memory Manager (Mm)**เป็นตัวควบคุม page table ซึ่งกำหนด mappingของvirtual address ไปสู่ physical memory
- **Executive** (Ex) เป็นส่วนของheap management และ synchronization
- **Object Manager** (**Ob**) เป็นศูนย์กลางควบคุม data object จำนวนมากในการทำงานของ Window NT ซึ่ง WDM driver ไว้ใจ Object Managerเท่านั้นสำหรับที่จะเก็บจำนวนที่ไว้อ้างอิง เพื่อป้องกัน objectจากการสูญหาย
- **Security reference Monitor (Se)** จะยอมให้ทำการcheck ที่ File system driver หรือไม่ บางครั้งในเรื่องของความปลอดภัยในเวลาที่ I/O request ไปที่ WDM driver

- **Run-time library (Rtl)** เป็นส่วนที่เก็บutility เช่น list และ string manager routine ซึ่งkernel mode สามารถใช้แทน ANSI standard library ส่วนใหญ่การทำงานในส่วนนี้จะดูได้จากชื่อของมัน
- Windows NT implement user mode ไปเรียก Win32 subsystemในkernel mode เกี่ยวกับ routine ที่มีชื่อนำหน้าว่า Zw ซึ่งประกอบไปด้วยfunctionที่เอาไว้สำหรับ access file และ registry
- **Windows NT ke**rnel (Ke) อยู่ทุกที่ในlow – level synchronizationของ activities ระหว่าง threads และ processors
- Layerล่างสุดของoperating system อยู่ที่**hardware abstraction layer** (HAL) HAL รู้ถึงวิธีการทำงานของinterrupt ,spinlock,address I/O และmemory map device ดังนั้นแทนที่จะพูดคุยโดยตรงกับhardwareนั้น WDM driverจะไปเรียกfunctionในHAL ขึ้นมาทำแทน

2.2.2.Side Effect

เราจะมาดูกันถึงในเรื่องของ Side Effect ว่ามีผลยังไงต่อโปรแกรมที่เราจะเขียนซึ่งบางคนอาจจะยังไม่รู้ถึงผลกระทบที่จะได้ในการประกาศหรือเขียนโปรแกรม ดังตัวอย่างที่อยู่ด้านล่างนี้

```
int a = 2, b = 42, c;
c = min(a++, b);
```

จากโปรแกรมด้านบนค่าของ a และ c จะมีค่าเท่าไหร

```
#define min(x,y) ((x) < (y) ? (x) : (y))
```

ถ้าคุณเอา a++มาแทน x คุณเห็นว่า a มีค่าเท่ากับ 4 เพราะ a++ได้รับการทำไป2ครั้ง แต่ค่าminจะเท่ากับ 3 ซึ่งคุณควรจะทำตามกฎเมื่อคุณเขียนโปรแกรม WDM driver

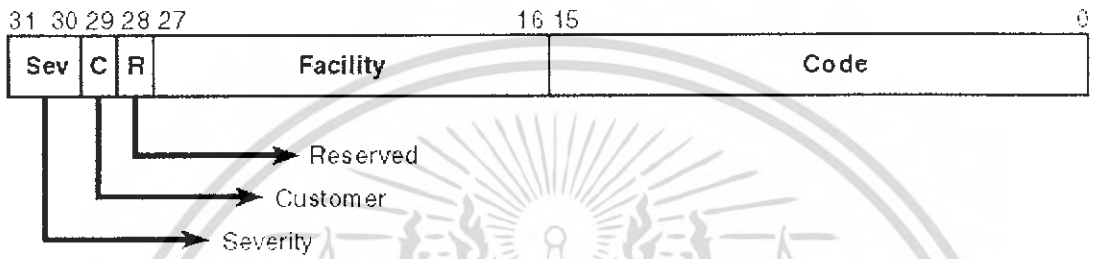
2.2.3.Error Handling

ความผิดพลาดส่วนใหญ่จะเกิดขึ้นจากมนุษย์เป็นคนทำ ซึ่งการทำงานในOperating System นั้นห้ามเกิดความผิดพลาด เพราะถ้าเกิดความผิดพลาดจะทำให้ระบบนั้นล่มหรือเกิดความเสียหายในส่วนต่างๆ ได้

ดังนั้นเราจึงทำการป้องกันความผิดพลาด โดยทั่วไป kernel mode สนับสนุน routineที่มีการ return status code ในการที่ดักจับความผิดพลาด โดยการรายงานความผิดพลาดนั้นอาจจะอยู่ในรูปแบบของ Boolean หรือ numerice value

2.2.4.Status code

Kernel mode มีการแสดงว่า success หรือ failure โดยการ return status code ไปที่ผู้เรียก ซึ่ง NTSTATUS value เป็นinteger ขนาด 32 bit ประกอบด้วย subfield ต่างๆดังรูปที่3.5 บิตที่อยู่สูงสุด 2บิต ใช้แสดงว่าreport นั้น success , information ,warning หรือ error ต่อมาเรามาดูที่customer bit ใช้สำหรับการเปลี่ยนแปลงหรือแก้ไขของเครื่องmainframe operating system ซึ่งเราไม่คิดว่าปัจจุบันจะยังได้ใช้อยู่ ต่อมาเรามาดูที่ facility code แสดงให้เห็นถึง system component ดังนั้นส่วนที่เหลือคือส่วนของ status code อีก 16 บิต แสดงเงื่อนไขของ report



รูปที่2.7 แสดงของ NTSTATUS code

คุณควรเช็ค status ที่return มาจาก routineนั้น จะไปหยุดหน้าที่การทำงานที่ทำอยู่เป็นประจำของ code นั้น ถ้าhigh bit ของstatus code เป็น 0 แล้วส่วนที่เหลือเป็นจำนวนใดๆแสดงว่า success

```
NTSTATUS status = SomeFunction(...);
if (!NT_SUCCESS(status))
  {
  <handle error>
  }
```

ตัวอย่าง การขั้นตอน initial AddDevice function กับ error checking

```
NTSTATUS AddDevice(PDRIVER_OBJECT DriverObject, PDEVICE_OBJECT pdo)
  {
  NTSTATUS status;
  PDEVICE_OBJECT fdo;
  status = IoCreateDevice(DriverObject, sizeof(DEVICE_EXTENSION),
    NULL, FILE_DEVICE_UNKNOWN, 0, FALSE, &fdo);
1 →  if (!NT_SUCCESS(status))
    {
2 →  KdPrint(("IoCreateDevice failed - %X\n", status));
    return status;
    }
  PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;
  pdx->DeviceObject = fdo;
  pdx->Pdo = pdo;
  pdx->state = STOPPED;
3 →  IoInitializeRemoveLock(&pdx->RemoveLock, 0, 0, 255);
  status = IoRegisterDeviceInterface(pdo, &GUID_SIMPLE, NULL,
    &pdx->ifname);
4 →  if (!NT_SUCCESS(status))
    {
    KdPrint(("IoRegisterDeviceInterface failed - %X\n", status));
    IoDeleteDevice(fdo);
    return status;
    }
    ...
  }
```

1. ถ้า IoCreateDevice fails เราจะreturn status code กลับไปให้มัน
2. เป็นการdebuging ตัวdriverโดยให้มันแสดง errorที่มันพบ
3. IoInitializeRemoveLock เป็นVOID function ซึ่งหมายความว่ามันจะไม่สามารถที่0tfailได้ ดังนั้นเราไม่จำเป็นต้องcheck status code
4. ถ้า IoRegisterDeviceInterface เกิด fail เราจะมีการcleanup ก่อนที่จะreturnไปให้ผู้ที่เรียกมัน โดยที่คุณจะต้องเรียกใช้ IodeleteDevice ให้การทำลายdevice object ที่เพิ่งสร้างมา

2.2.5. Memory Management

ในส่วนนี้จะอธิบายถึงmemory management ซึ่ง windows2000มีการแบ่ง virtual address space อยู่หลากหลายวิธี

windows2000มีวิธีใช้ memory management อยู่หลายวิธีโดยเราจะพูดถึง2วิธีพื้นฐานคือ ExAllocatePool และ ExFreePool ให้คุณไว้ใช้สำหรับ allocate และ release ขนาดของblockจาก heap เราจะอธิบายอย่างคร่าวๆว่าคุณจะใช้ memory blockไปไว้ในlink listsของ structure สุดท้ายเราจะอธิบาย concept ของ lookaside list ที่ซึ่งอนุญาติคุณให้ allocate และ release block นั้น

How Big Is a Page?

ใน virtual memory system , physical memory และ swap file ไปสู่ page frame ในWDM driver คุณสามารถใช้ PAGE_SIZE บอกว่าคุณมีpageขนาดเท่าไหร ซึ่งใน Windows NTนั้น page มีขนาด 4096 byte

```
PAGE_SIZE == 1 << PAGE_SHIFT
```

คุณสามารถใช้macrosในการทำงานเกี่ยวกับขนาดของpage

-ROUND_TO_PAGES

-BYTES_TO_PAGES

-BYTE_OFFSET

-PAGE_ALIGN

-ADDRESS_AND_SIZE_TO_SPAN_PAGES

2.2.6.Synchronization

Microsoft Windows 2000เป็น multitasking operation system ซึ่งสามารถทำงานในsymmetric multiprocessor environment ซึ่งเราจำเป็นที่จะต้องรู้และเข้าใจผู้เขียนdriverซึ่งจะต้องนำcodeไปexecute ในcontext ของ theadๆหนึ่งและในภาวะฉุกเฉินของ multitasking สามารถควบคุมจาก การทำงานของmomentใดๆ ยิ่งกว่านั้นการที่executionของmultiple thread เป็นกลไกบนmultiprocessor ซึ่งwindowsอนุญาติให้คุณแก้ไขปัญหาโดยการใช้interrupt request level (IRQL) priority scheme และ การอ้างถึง และ แก้spin lock ซึ่งcodeในส่วนนี้ IRQL หลีกเลี่ยงการทำลายpreemtionบนsingle cpu

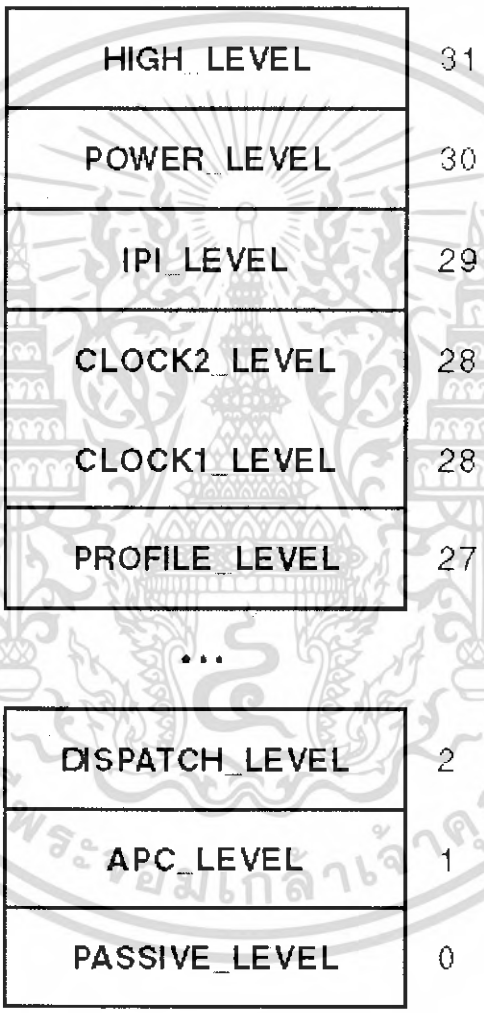
2.2.7.Interrupt Request Level

Windows NT มีการกำหนด priority level ที่รู้จักกันในชื่อ interrupt requestซึ่ง IRQL จะจัดเตรียม วิธีการ synchronization สำหรับการทำงานบน single CPU โดยมีกฎดังนี้:

CPU หนึ่งจะรัน IRQL บน PASSIVE_LEVEL ได้ การทำงานบน CPU สามารถ preempt ได้โดย การทำงาน execute ของตัวที่มี IRQL สูงกว่า

รูปที่2.8 จะแสดงการจัดลำดับค่าของ IRQL สำหรับ x86 platform โปรแกรมที่รันบน User-mode จะรัน PASSIVE_LEVEL เพราะฉะนั้นการ preempt จะทำการ execute และเลื่อนลำดับ IRQL ให้สูงขึ้น มีฟังก์ชันมากมายที่ถูก execute ที่ PASSIVE_LEVEL โดย DriverEntry และ AddDevice ได้อธิบายไว้ใน บทที่ 2 เรื่อง “Basic Structure of a WDM Driver” ส่วนในบทนี้จะเป็น I/O request packet (IRP) เสีย ส่วนใหญ่

Driver routine พื้นฐานจะ execute ที่ Dispatch_Level ซึ่งสูงกว่า PASSIVE_LEVEL จะมีการรวม StartIo, deferred procedure call (DPC) routines และฟังก์ชันอื่นๆ เข้าไปด้วย ในส่วนของพื้นฐานที่ต้องการติดต่อพื้นที่ส่วนของ device object และ device extension ที่ปราศจากการแทรกแซงจาก driver dispatch routine เมื่อ routines นี้กำลังรันอยู่จะรับประกันว่าจะไม่มี thread ที่สามารถ preempt การ execute จาก driver dispatch routine ได้เพราะ dispatch routine รันที่ IRQL ที่ต่ำกว่า นอกจากนี้ไม่มี thread ที่สามารถ preempt ได้เพราะรันอยู่ใน IRQL เดียวกัน การที่จะ preempt ได้จะต้องถูก preempt จาก thread ที่มี IRQL สูงกว่าเท่านั้น



รูปที่2.8 Interrupt request level

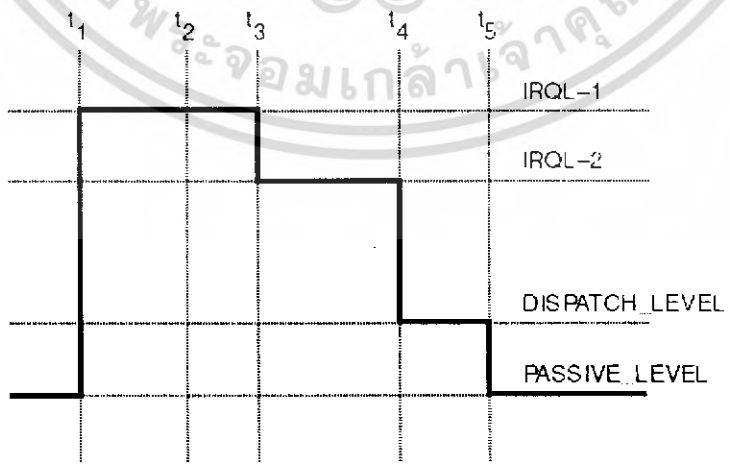
ระหว่าง DISPATCH_LEVEL และ PROFILE_LEVEL คือห้องสำหรับ interrupt level ของอุปกรณ์ภายนอกต่างๆ แต่ละอุปกรณ์จะมีการสร้าง interrupt ที่มี IRQL ที่มีกำหนดลำดับความสำคัญเมื่อมาเปรียบเทียบกับอุปกรณ์อื่น WDM มีการจัดการ IRQL สำหรับ interrupt ของมันเมื่อมันได้รับ

IRP_MJ_PNP request ด้วยฟังก์ชัน minor code IRP_MN_START_DEVICE. Interrupt level ของอุปกรณ์มีการกำหนดข่าวสารสำหรับส่งไปให้ตัวแปรของการร้องขอนั้นๆ เราจะอ้างถึงอุปกรณ์ IRQL โดยเรียกสั้นๆว่า DIRQL ซึ่งมันไม่ใช่ single request level ในทางตรงกันข้ามมันเป็น IRQL สำหรับการร้องขอเชื่อมความสัมพันธ์กับอุปกรณ์ต่างๆ ซึ่งจะอธิบายในหัวข้อต่อไป

IRQL level อื่นๆ มีจุดประสงค์บางอย่างแล้วแต่สถาปัตยกรรม CPU นั้นๆ ตั้งแต่ level ถูกนำมาใช้ใน Windows NT kernel มันไม่มีจุดประสงค์พิเศษในการเชื่อมความสัมพันธ์งานการเขียน device driver จุดประสงค์ของ APC_LEVEL สำหรับตัวอย่างคือยอมให้ระบบกำหนด asynchronous procedure call (APC) ซึ่งจะอธิบายรายละเอียดที่ตอนท้ายของบทนี้ สำหรับ thread จะปราศจากการแทรกแซงจาก thread อื่นๆภายใน CPU เดียวกัน การทำงานจะเกิดขึ้นที่ HIGH_LEVEL

2.2.8.IRQL in Operation

จะแสดงให้เห็นถึงความสำคัญของ IRQL ซึ่งแสดงในรูปที่2.9 ซึ่งแสดงให้เห็นถึงความเป็นไปได้ของเหตุการณ์บน Single CPU เมื่อการเริ่มต้นของลำดับ CPU จะทำการ execute ที่ PASSIVE_LEVEL ที่เวลา t1 การขัดจังหวะมาถึงบริการ routine execute ที่เวลา IRQL-1 ระดับหนึ่งระหว่าง DISPATCH_LEVEL และ PROFILE_LEVEL ที่เวลา t2 มาถึงการขัดจังหวะของบริการ routine execute ที่ IRQL-2 ซึ่งน้อยกว่า IRQL-1 เพราะกฎการ preempt แสดงถึงการบริการต่อเนื่องของ CPU เมื่อการขัดจังหวะครั้งแรกของบริการ routine เสร็จสิ้นที่เวลา t3 มันอาจจะส่ง request DPC ซึ่ง DPC routine จะ execute ที่ DISPATCH_LEVEL ผลที่ตามมาเมื่อตัวที่มีลำดับความสำคัญสูงกว่ากำลังทำงานบริการ routine เพื่อการขัดจังหวะครั้งที่ 2 เพราะฉะนั้นการ execute ถัดไป เมื่อเวลา t4 เสร็จสิ้น การกำหนดไม่สามารถทำให้มีอะไรเกิดขึ้นในเวลาเฉลี่ย DPC จะรันที่ DISPATCH_LEVEL เมื่อ DPC routine เสร็จสิ้นที่เวลา t5 IRQL จะสามารถหยุดอยู่ที่ PASSIVE_LEVEL ได้



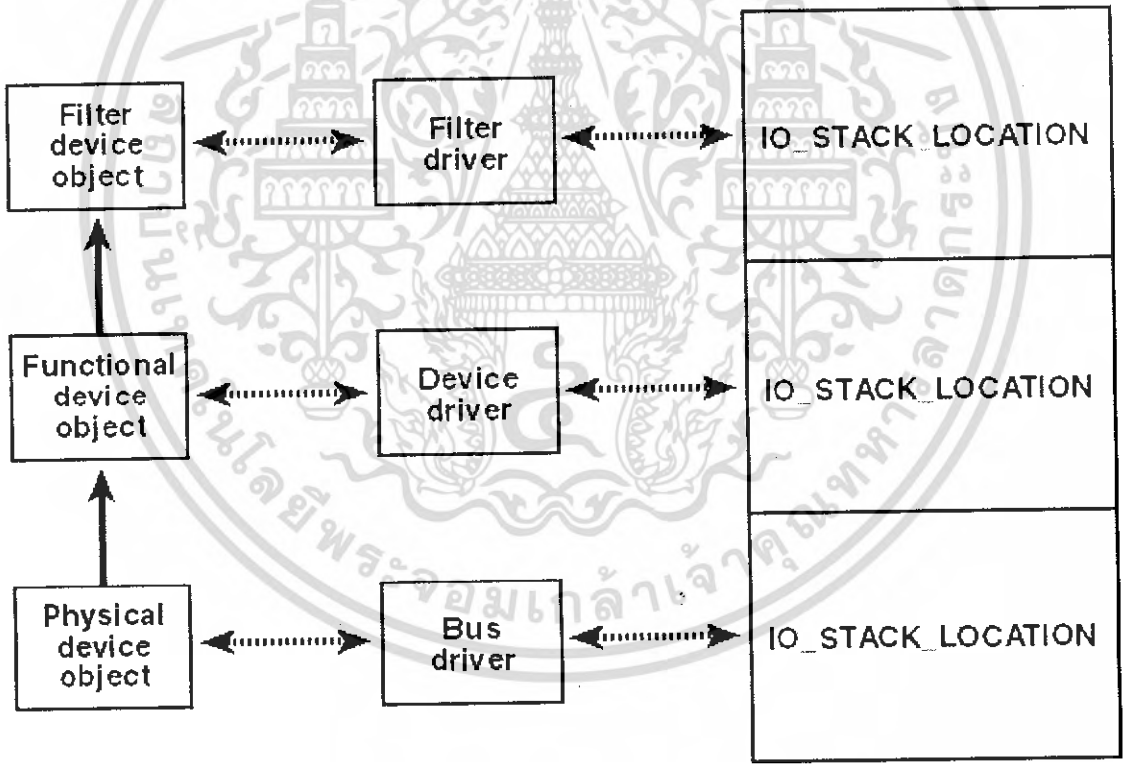
รูปที่2.9 Interrupt priority

2.2.9.The I/O Request Packet

Operating system ใช้data structure ที่เรียกว่า I/O request packet หรือ IRP ในการติดต่อกับ kernel-mode ในบทนี้ได้อธิบายเกี่ยวกับส่วนที่สำคัญของdata structure และโดยหมายถึง create , sent ,process และ ultimately destroy สุดท้ายเราจะอธิบายถึงความสัมพันธ์ของส่วนประกอบต่างๆของแนวคิดซึ่งสิ่งแวดล้อมจะเป็นตัวกำหนดชนิดของIRP เพราะฉะนั้นคุณอาจต้องการความรู้ในบทนี้และอ้างอิงที่ซึ่งคุณกำลังอ่านในบทนี้

2.2.10.The I/O Stack

เมื่อไหร่ก็ตามที่ทุกๆkernel-mode สร้างIRP มันจะสร้างความสัมพันธ์ด้วยarrayของโครงสร้าง IO_STACK_LOCATIONโดยที่ตำแหน่งของstackตำแหน่งหนึ่งสำหรับแต่ละdriverซึ่งprocessเป็น IRPและตำแหน่งของstackเดี่ยวๆเป็นประจำสำหรับใช้ของIRPตัวเดิม(ดูรูปที่ 2.10) ตำแหน่งstackเก็บชนิดของcode และ parameter สำหรับIRP ด้วยaddressของroutineที่ทำเสร็จแล้ว



รูปที่2.10 parallelismระหว่างdriverและI/O stack

2.2.11. The DeviceIoControl API

เป็นขั้นตอนการใช้งานฟังก์ชันมาตรฐานต่างๆ ของ Microsoft Win32 ซึ่งเป็นฟังก์ชันที่ทำงานบน user-mode driver การทำงานของฟังก์ชันจะทำการติดต่อกับอุปกรณ์ฮาร์ดแวร์ ผ่าน kerner-mode driver โดยจะส่ง IRP ไปบอกให้ kernel-mode driver ทราบว่าตอนนี้ user-mode driver ต้องการทำอะไร รายละเอียดของฟังก์ชันต่างๆ มีดังนี้

2.2.11.1 ฟังก์ชัน DeviceIoControl()

เป็นการเรียกใช้ฟังก์ชันมาตรฐาน ของ Microsoft Win32 ซึ่งเป็นฟังก์ชันที่ทำงานบน user-mode driver และการทำงานจะส่ง IRP_MJ_DEVICE_CONTROL ไปให้กับ kernel-mode driver เพื่อทำการติดต่อกับอุปกรณ์ฮาร์ดแวร์ โดยฟังก์ชันที่ใช้ในการงานติดต่อกับอุปกรณ์ฮาร์ดแวร์นี้คือฟังก์ชัน DeviceIoControl มีรูปแบบการใช้งานดังนี้

```
result = DeviceIoControl(Handle, Code, InputData, InputLength,
  OutputData, OutputLength, &Feedback, &Overlapped);
```

ค่าที่ได้รับจากการเรียกใช้ฟังก์ชันนี้คือ ค่า Boonlean ถ้าสามารถติดต่อกับ kernel-mode driver หรือว่าภายในระบบปฏิบัติการวินโดว์มี device driver ที่สามารถตอบสนองต่อ IOControl Code นี้ได้ก็จะส่งค่ากลับเป็น TRUE ถ้าไม่สามารถติดต่อกับ kernel-mode driver หรือว่าภายในระบบปฏิบัติการวินโดว์ไม่มี device driver ที่สามารถตอบสนองต่อ IOControl Code นี้ได้ก็จะส่งค่ากลับเป็น FALSE และค่า parameter ที่ส่งไปมีทั้งหมด 8 ตัว แต่มีตัวหลัก ๆ เพียง 6 ตัวคือ

ค่า parameter ตัวที่ 1 เป็นค่า Handle ของอุปกรณ์ฮาร์ดแวร์ที่เราต้องการจะติดต่อด้วย ได้รับมาจากการใช้ฟังก์ชัน CreateFile ซึ่งจะอธิบายในหัวข้อถัดไป

ค่า parameter ตัวที่ 2 เป็นค่า IOControl Code ที่โปรแกรม user-mode driver ส่งไปให้กับ kernel-mode driver ซึ่งค่า IOControl Code นี้จะมีการประกาศค่าไว้ใน kernel-mode driver และโปรแกรม user-mode driver ที่ต้องการจะติดต่อด้วย ต้องส่งค่า IOControl Code ที่ตรงกับที่ได้ประกาศไว้ใน kernel-mode driver ด้วย ถ้าหากมีการส่งค่าที่ไม่ตรงกับที่ประกาศไว้ใน kernel-mode driver ก็จะไม่สามารถติดต่อกับอุปกรณ์ฮาร์ดแวร์ได้ และจะมีการส่งค่ากลับมาที่ฟังก์ชันเป็น FALSE

ค่า parameter ตัวที่ 3 เป็นค่าข้อมูลที่ต้องการะส่งไปให้กับ kernel-mode driver เพื่อดิดต่อหรือส่งไปเพื่อสั่งงานอุปกรณ์ฮาร์ดแวร์ โดยข้อมูลที่ส่งไปจะเป็นการส่งค่าพื้นที่ในหน่วยความจำที่เก็บค่าข้อมูลนั้นไว้

ค่า parameter ตัวที่ 4 เป็นค่าขนาดของข้อมูลที่ทำการส่งไปให้ kernel-mode driver โดยส่วนมากค่านี้จะเป็นค่าตัวเลข

ค่า parameter ตัวที่ 5 เป็นค่าของพื้นที่ในหน่วยความจำที่ทำการส่งไปเพื่อใช้ในการรับค่าของข้อมูลที่จะส่งกลับมาจาก kernel-mode driver หรือเป็นค่าที่ส่งกลับมาจากอุปกรณ์ฮาร์ดแวร์ที่ส่งค่ากลับมาให้กับ user-mode driver โดยส่งผ่าน kernel-mode driver

ค่า parameter ตัวที่ 6 เป็นค่าขนาดของข้อมูลที่ได้รับกลับมาจาก kernel-mode driver โดยส่วนมากค่านี้จะเป็นค่าตัวเลข

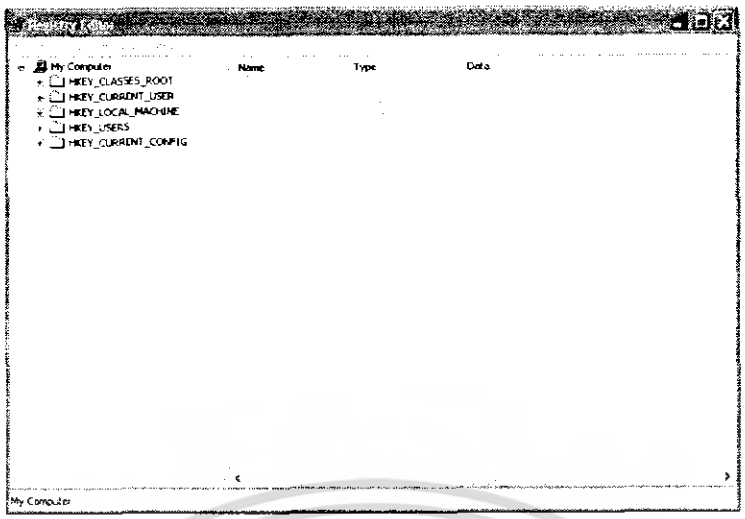
2.2.11.2 ฟังก์ชัน CreateFile()

เป็นการเรียกใช้ฟังก์ชันมาตรฐาน ของ Microsoft Win32 ซึ่งเป็นฟังก์ชันที่ทำงานบน user-mode driver และการทำงานจะส่ง IRP_MJ_CREATE ไปให้กับ kernel-mode driver เพื่อทำการสร้างไฟล์ Handle ของอุปกรณ์ฮาร์ดแวร์ เพื่อไว้สำหรับให้ user-mode driver ใช้ในการอ้างอิงถึงอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อด้วย รูปแบบการใช้งานมีดังนี้

```
Handle = CreateFile("\\\\.\\IOCTL", GENERIC_READ | GENERIC_WRITE,
  0, NULL, OPEN_EXISTING, flags, NULL);
if (Handle == INVALID_HANDLE_VALUE)
  <error>
...
CloseHandle(Handle);
```

ค่าที่ได้รับมากลับมาจาฟังก์ชันนี้คือค่า Handle ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อด้วย ถ้าหากสามารถทำการติดต่อกับอุปกรณ์ฮาร์ดแวร์ได้ก็จะได้รับค่า Handle ของอุปกรณ์นั้นกลับมา แต่ถ้าหากไม่สามารถติดต่อกับอุปกรณ์ฮาร์ดแวร์ได้ค่าที่จะได้รับกลับมาก็คือค่า INVALID_HANDLE_VALUE และค่าที่ทำการส่งไปมีทั้งหมด 7 ตัว แต่ค่าหลักๆ ที่จำเป็นต้องรู้คือ

ค่า parameter ตัวที่ 1 คือชื่อของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อด้วย ในที่นี้คือ IOCTL และพาธของการ Create File คือ \\.\ จะเป็ นรูทของระบบเช่น C:\windows\system32 ระบบปฏิบัติการจะทำการแปลงเป็น \device\ HarddiskVolume1\Partition0 แต่เราไม่สามารถที่รู้ได้ว่าผู้ใช้ทั่วไปลงระบบปฏิบัติไว้ที่ใด จึงค่าเป็นรูทของระบบซึ่งจะมี Symboliclink ที่จะทำการเชื่อมโยงไปยัง Device name ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อได้ หากเราจะต้องการตรวจสอบว่า Device name ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อของเราได้ใส่เข้าไปในระบบหรือยัง ก็สามารถทำการตรวจสอบได้โดยการเข้าไปดูที่ Registry ของระบบปฏิบัติการ Windows เข้าไปที่ Run... พิมพ์คำว่า regedit กด OK จากนั้นเราก็จะเข้าสู่หน้าต่าง Registry Editor ของระบบปฏิบัติการ Windows ดังรูปที่ 2.11

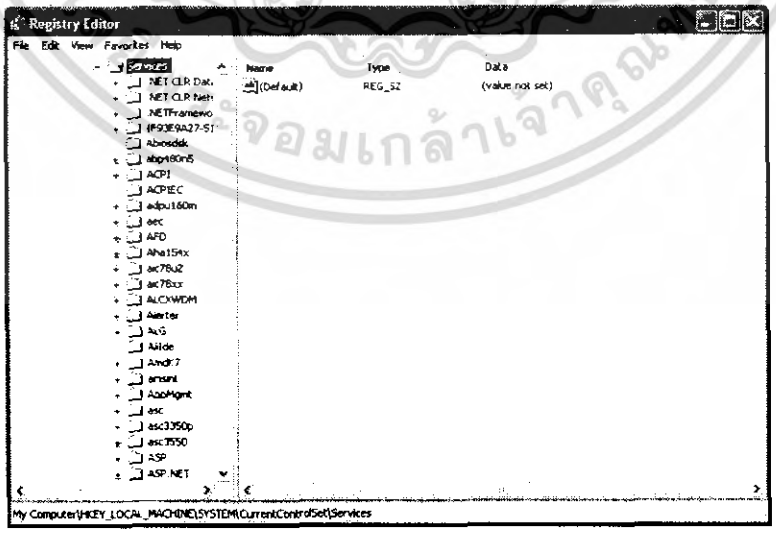


รูปที่ 2.11 หน้าต่าง Registry Editor

หากเราจะต้องการตรวจสอบว่า Device name ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อของเราได้ใส่เข้าไปในระบบหรือยัง ให้เข้าไปดูที่พาธ

RegistryPath = HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

ดังรูปที่ 2.12 จะแสดง Device name ของอุปกรณ์ฮาร์ดแวร์ ที่มีอยู่ภายในเครื่อง ที่พาธ HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services ซึ่งจะมี Device name ของอุปกรณ์ฮาร์ดแวร์หลายชนิด ที่ได้ทำการติดตั้ง Device Driver ลงไปในเครื่องระบบปฏิบัติการ Windows



รูปที่ 2.12 พาธ HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

ค่า parameter ตัวที่ 2 เป็นการบอกกับระบบปฏิบัติการ windows ว่าต้องเปิดไฟล์ Handle นี้เพื่อที่จะอ่านข้อมูลจากอุปกรณ์ฮาร์ดแวร์หรือเพื่อที่จะเขียนข้อมูลยังอุปกรณ์ฮาร์ดแวร์ ในที่นี้ใช้เปิดเพื่อทั้งอ่านและเขียน จึงใช้ operater OR เข้ามาช่วย

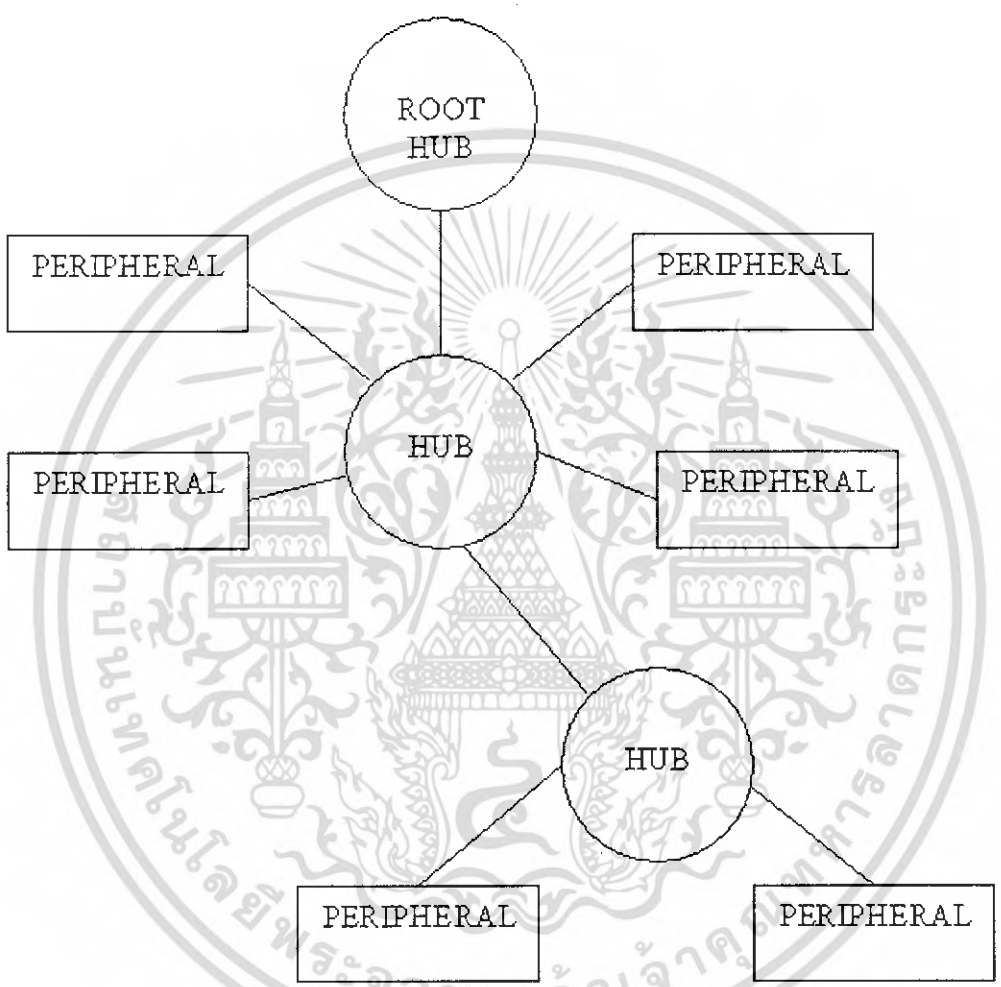
ส่วนถัดมาจากการ CreateFile() คือการตรวจสอบว่าสามารถสร้างไฟล์ Handle ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อได้หรือไม่ ถ้าหากสามารถทำการติดต่อกับอุปกรณ์ฮาร์ดแวร์ได้ก็จะได้รับค่า Handle ของอุปกรณ์ฮาร์ดแวร์นั้นกลับมา แต่ถ้าหากไม่สามารถติดต่อกับอุปกรณ์ฮาร์ดแวร์ได้ค่าที่จะได้รับกลับมาก็คือค่า INVALID_HANDLE_VALUE เราสามารถใช้ if ตรวจเช็คค่าได้

เมื่อทำการ CreateFile() ได้สำเร็จ ก็ทำการรับ – ส่งข้อมูลกับอุปกรณ์ฮาร์ดแวร์ตามปกติ โดยใช้ไฟล์ Handle ที่ได้จากการ CreateFile() ในการอ้างอิงกับอุปกรณ์ฮาร์ดแวร์ที่จะติดต่อด้วย จากนั้นเมื่อการทำงานเสร็จสิ้นก็จะใช้ฟังก์ชัน CloseHandle() ในการคืนค่าไฟล์ Handle ให้กับระบบโดยค่า parameter ที่ส่งไปก็คือค่าไฟล์ Handle ที่ได้สร้างขึ้นมาจากฟังก์ชัน CreateFile ในตอนแรก

2.3.ทฤษฎีเบื้องต้นส่วนของ USB

2.3.1 Bus topology

Topologyบนbusจากรูปจะเห็นได้ว่าที่จุดกลางของแต่ละstarจะเป็นhub ซึ่งแต่ละจุดบน star เป็น device ที่ต่ออยู่กับhub ซึ่งจำนวนของpoint บนแต่ละ star เปลี่ยนแปลงได้ โดยเราสามารถเพิ่มจำนวนport ได้ ซึ่งทุกdevices บน bus จะต้องแบ่งกันใช้ data path ที่จะใช้เพราะมีอยู่เส้นเดียวเท่านั้น



รูปที่ 2.13 USB ใช้ Topology แบบ Star มีHub เป็นศูนย์กลาง

2.3.2.Host Duties

2.3.2.1.Detect Device

เมื่อเราเปิดเครื่อง hub จะบอกให้ host ทราบว่ามี Device ติดต่ออยู่ซึ่งในprocess นี้เรียกว่า การenumurate โดยที่ host จะ assign address และ request ข้อมูลจากแต่ละ device

2.3.2.2.Manage data flow

Host จะเป็นตัวจัดการflowของdata เมื่ออุปกรณ์หลายตัวต้องการที่จะ transferข้อมูลในเวลาเดียวกัน host controllerจะจัดการโดยแบ่ง data path ละ 1 millisecond ให้แต่ละport ของแต่ละ frame

2.3.2.3.Error Checking

Host มีหน้าที่ในการcheck error มันเพิ่ม error checking bit ลงไปใน data ที่ส่งไป เมื่อ device driver ได้รับdata มันจะสามารถประมวลผลและเปรียบเทียบ ถ้าผลลัพธ์ที่ได้mathกัน device จะไม่ ack ข้อมูลที่รับเข้ามาและ host จะรู้ว่าควรทำการส่งซ้ำอีกครั้ง

2.3.3.Peripheral Duties

ในหลากหลายวิธี peripheral หรือ device มีหน้าที่เป็นเหมือนกระจกที่สะท้อน host เมื่อ host นั้น initial การติดต่อกับ peripheral นั้นต้อง respond กับ deviceไม่สามารถติดต่อกับ usb ด้วยตัวมันเอง มันต้องรอและ respond เมื่อ host ติดต่อมา ซึ่ง USB Controller ในdevice จะเป็นตัวจัดการของตัวมันเอโดยอัตโนมัติ

2.3.3.1.Detect Communication

แต่ละ device จะคอยจับตาดู device address ในแต่ละการติดต่อบนbus ถ้าaddressไม่mathกับ address ที่เก็บไว้ มันจะไม่สนใจ แต่ถ้า address นั้น mathกัน deviceจะเก็บdataลงในbuffer และ generate interrupt สัญญาณไปบอกว่า data ส่งมาถึงแล้ว

2.3.3.2.Respond

เมื่อเปิดเครื่องหรือเมื่อdevice attaches กับ power system device ต้อง respond ต่อ request ที่ host สร้างใน enumerate process ซึ่ง USB ทุกตัวต้อง respond

2.3.4.Device Endpoints

การส่งข้อมูลทั้งหมดจะเดินทางไปหรือมาจาก device endpoint ซึ่งendpoint เป็น bufferซึ่งเก็บข้อมูล โดยทั่วไปมันจะเป็นblockของข้อมูล หรือ register ที่อยู่ใน chip controller ซึ่งข้อมูลที่ถูกเก็บไว้ที่ endpoint อาจเป็นข้อมูลที่รับมา หรือ ข้อมูลที่รอส่ง โดยที่ host มีbufferสำหรับรับข้อมูลและข้อมูลที่รอพร้อมจะส่ง แต่hostไม่มีendpoint แทนที่มันจะทำงานเป็นจุดเริ่มสำหรับการติดต่อกับdevice endpoint

รายละเอียดที่กำหนดdevice endpoint ซึ่ง addressของportจะเป็นaddressที่ไม่ซ้ำกับใครเอาไว้ติดต่อระหว่าง host กับ device โดยที่addressของแต่ละ endpointประกอบด้วย endpoint number และ direction

โดยnumberอาจอยู่ในระหว่าง 00-0Fh. ส่วนdirection เป็นภาพที่เรามองจากhost ซึ่งจะมี IN และ OUT โดยที่ IN จะเป็นทิศทางที่เข้าสู่ host และ OUTจะเป็นทิศทางที่ออกจาก host เสมอ ที่endpoint จะถูกกำหนดโดยใช้ control transfer ในการส่งข้อมูลทั้ง INและOUT ซึ่งต้องแบ่งendpoint number

ทุกdevice ต้องมี Endpoint 0 ไว้กำหนดendpoint ซึ่งการส่งข้อมูลแบบอื่นจะเป็นแบบทิศทางเดียวเท่านั้น endpointหนึ่งสามารถเป็นได้ทั้ง INและOUT ตัวอย่างเช่น endpoint 1 สามารถเป็น IN endpoint address พร้อมกับเป็นOUT endpoint address

2.3.5.Pipes

ก่อนจะสามารถเกิดการส่งข้อมูลได้นั้น hostและ deviceต้องสร้างpipe ขึ้นมาเสียก่อน pipe ไม่ได้เป็น physical object แต่มันเป็นการรวมกันระหว่าง device endpointและ host controller software

Host จะสร้างpipe หลังจากที่ระบบถูกบูทขึ้นมา หรือ device มีการทำ attachment ถ้าdeviceถูกนำเอาออกไปจากบัส hostจะนำเอาpipe ที่ไม่จำเป็นออกตามไปด้วย หรือ host อาจจะไปร้องขอให้สร้าง pipe ตัวใหม่ขึ้นมา หรือเอาออกในเวลาต่อมาก็ได้

รายละเอียดของข้อมูลที่ได้รับที่host จะประกอบด้วยdescriptor สำหรับแต่ละendpoint ซึ่งแต่ละendpoint descriptor จะเป็น blockของข้อมูลซึ่งบอกhostว่า อะไรที่มันจำเป็นต้องรู้เกี่ยวกับendpointในการติดต่อกับมัน ซึ่งจะประกอบไปด้วย endpoint address ,ชนิดของการส่งข้อมูล และขนาดของpacket สูงสุด

ในบางกรณี hostจะสร้างpipeเดียวเท่านั้นหลังจากที่แน่ใจว่าbusมีbandwidthพอที่จะทากรส่งข้อมูลกายในเวลาที่requestมา กรณีนี้จะทำเมื่อเราใช้ isochronous transfer

ในบางกรณีhostจะตรวจสอบ bandwidthดูก่อนสร้าง pipe ถ้าbandwidthพอกับที่ต้องการ hostจะรับrequestและทำการส่งข้อมูลแต่ถ้า bandwidth ไม่พอ hostจะปฏิเสธrequestที่จะสร้างpipe และ software ต้องrequestเข้ามาใหม่อีกครั้งหนึ่ง หรือ ไม่ก็จะรอจนกว่าbandwidthจะพอกับที่ต้องการ

2.3.6.ชนิดของการส่งข้อมูล

USBนั้นถูกออกแบบให้มีhandleอยู่หลายชนิดในการจัดการกับ periperals ที่มีความต้องการสำหรับ transfer rate , response time และ error correcting ซึ่งทั้ง4 ชนิดของการส่งข้อมูลแต่ละความจำเป็นที่แตกต่างกันและอุปกรณ์ที่รองรับกับชนิดของการส่งข้อมูลซึ่งต้องเหมาะสมกับวัตถุประสงค์ ซึ่งสามารถแบ่งออกเป็น4 ชนิดดังนี้

2.3.6.1.Control transfers

Control transfers เป็นชนิดเดียวเท่านั้นที่มีfunctionในการกำหนดรายละเอียดของUSB การส่งข้อมูลแบบนี้นั้นจะทำให้host สามารถอ่านและเลือกรูปแบบและsetอย่างอื่นได้ในการenumerateซึ่งdeviceทุกตัวต้องรองรับcontrol transfer

2.3.6.2.Bulk transfers

Bulk transfers เป็นวิธีที่ใช้ในสถานการณ์ที่ไม่เกิดการคับคั่งของเส้นทาง อย่างเช่นการส่งไฟล์ไปที่เครื่องพิมพ์ หรือการรับข้อมูลจากเครื่องสแกนเนอร์ ในกรณีนี้เป็นวิธีที่มีการส่งข้อมูลได้รวดเร็ว แต่ข้อมูลสามารถถูกพักไว้ก่อนได้ถ้าจำเป็น

2.3.6.3.Interrupt transfers

Interrupt transfers ถูกนำไว้ใช้สำหรับ device ที่ต้องรับ attention อย่างรวดเร็วจาก host หรือ device ซึ่ง Interrupt transfers เป็นวิธีเดียวเท่านั้นที่สามารถใช้ใน low-speed device อย่างเช่น keyboard หรือ mouse ที่สามารถใช้ Interrupt transfers ในการส่ง keypress หรือ ตำแหน่งของ mouse ที่เคลื่อนที่ไป

2.3.6.4.Isochronous transfers

Isochronous transfers ถูกนำไว้ใช้สำหรับ deviceที่ต้องส่งข้อมูลที่อัตราคงที่ อย่างเช่น การส่งไฟล์เสียงเพื่อใช้การทำแบบreal time ซึ่งต้องมีอัตราคงที่

2.3.7.รูปแบบการส่งข้อมูลที่ใช้ในโปรแกรม

Bulk Transfers

Bulk Transfers ประกอบด้วยมากว่าหนึ่ง in หรือ out transaction ซึ่ง Bulk Transfers เป็นแบบ one – way ซึ่ง action ใน transactions ถ้าเป็น IN ก็ต้องเป็น IN ทั้งหมด หรือถ้าเป็น OUT ก็ต้องเป็น OUT ทั้งหมด

Data Size

Bulk Transfers สามารถมี Maximum Packet ได้ 8,16,32 ถึง 64 bytes ในระหว่างการ enumerate นั้น host จะดูว่ามี maximum packet เท่าไรจาก device ซึ่งข้อมูลนั้นอาจจะมีขนาดที่น้อยกว่าหรือเท่ากันหรืออาจจะมีขนาดใหญ่กว่า maximum ถ้ากลุ่มข้อมูลมีขนาดมากกว่า host controller จะแบ่งไปเป็น multiple transaction และ low overhead

2.3.8. Stream และ Message Pipes

ในการแบ่ง pipeออกเป็นชนิดของการส่งข้อมูล USBได้กำหนดpipeเป็น stream หรือ message type ซึ่ง control transferเป็นวิธีเดียวที่ใช้ message pipe ทั้งสองด้าน ส่วนวิธีอื่นจะเป็นstream pipe ทั้งหมด

ใน message pipe แต่ละครั้งจะเริ่มที่จะเริ่มที่ Setup transactionที่บรรจุ request มา เมื่อส่งเสร็จแล้ว host และ device อาจแลกเปลี่ยนข้อมูลและข้อมูลสถานะกัน ส่วนใน stream pipe นั้น

ข้อมูลจะไม่มีรูปแบบที่กำหนดตายตัว ซึ่ง device จะรับข้อมูลอะไรก็ตามที่ส่งมา แล้ว device firmware หรือ host จะเป็นคนประมวลผลข้อมูลนั้นเอง

2.3.9. Enumeration

ก่อนที่ applications จะสามารถติดต่อกับ deviceได้นั้น host จำเป็นต้องเรียนรู้เกี่ยวกับชนิดของการส่งข้อมูล และ endpoint ที่ device นั้นรองรับอยู่ ดังนั้นhostจะต้องกำหนด addressให้กับ device แล้ว host สามารถแลกเป็นข้อมูลกันได้ เรียกว่า enumeration

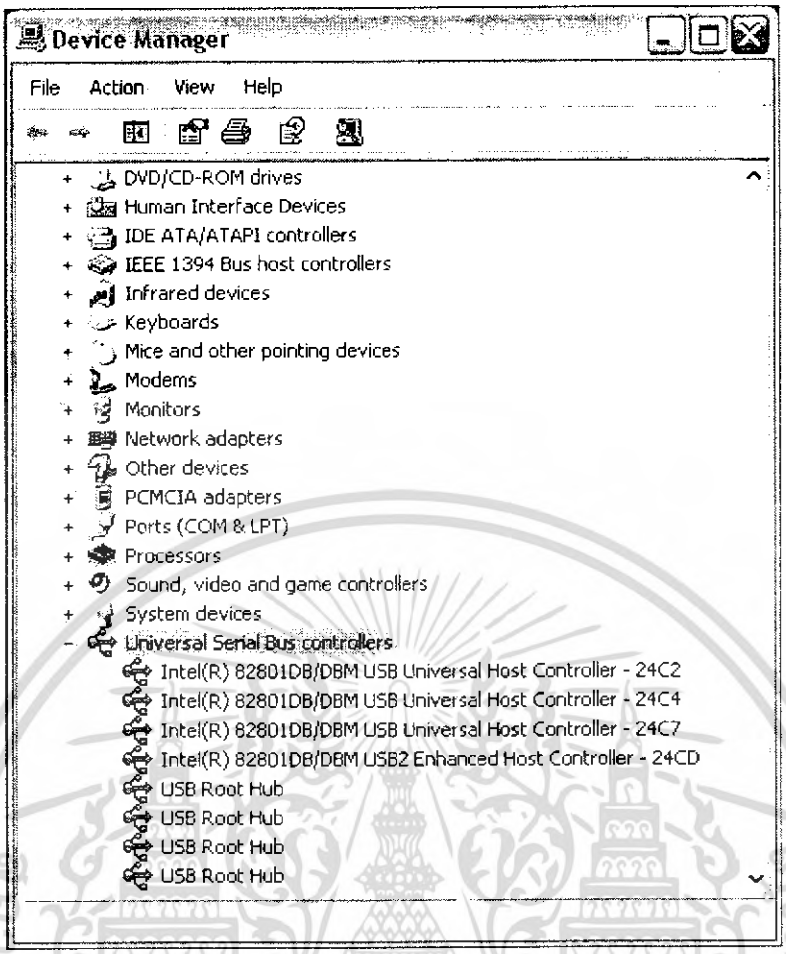
ระบบการทำงานของการ Enumeration อย่างแรกก็คือ หน้าที่ของ hub ที่จะเป็นคนตรวจหาว่ามี device ที่เข้ามาในระบบ หรือถูกนำออกไป ซึ่งในแต่ละ hub มี IN pipe ไว้สำหรับรายงานเหตุการณ์ไปที่ host เมื่อระบบถูกบูทขึ้นมา hostจะสำรวจ root hub ถ้ามีdeviceใดติดต่อเข้ามาไหม และจะทำอย่างนี้ต่อไปเป็นระยะๆ เพื่อจะดูว่ามีdevice ตัวไหนที่ถูกattachment หรือ removeออกไป

ในการเรียนรู้ device ตัวใหม่ host จะส่ง request ไปที่ device hub ทำให้ hub นั้นสร้างเส้นทางการติดต่อระหว่าง host และ device ซึ่ง host จะพยายาม ระบุ device โดยการที่จะ enumerate นั้นจะต้องกำหนดค่าเริ่มต้นแลกเปลี่ยนข้อมูลทำให้ host ได้ device driver ที่ติดต่อกับ device ได้ โดยการทำงานของมันนั้นจะประกอบไปด้วยการกำหนด address ของ device อ่านข้อมูลจากdevice กำหนดค่าและโหลด device driver ขึ้นมาและเลือกข้อกำหนดต่างที่ระบุไว้ ดังนั้น device ที่ถูกกำหนดและพร้อมที่จะ transfer
ข้อมูลโดยการใช้ endpointใดๆในที่เรากำหนด

Host จะenumerate โดยส่ง control transfer ที่ใช้endpoint 0 ซึ่งทุกdeviceจะต้องรองรับcontrol transferนี้ เมื่อenumerateเสร็จแล้วdeviceนั้นต้องตอบแต่ละrequestโดยreturnกลับไปหาและทำงานตามrequestนั้น

จากมุมมองของผู้ใช้นั้น ไม่ควรจะให้เห็นเกี่ยวกับการenumerate และควรทำงานอย่างอัตโนมัติ สำหรับในบางกรณีที่ windows บอกว่า ตรวจพบdeviceตัวใหม่ และในบางครั้งถ้าเป็นการใช้ครั้งแรก ผู้ใช้ควรจะเตรียมแผ่นซึ่งบรรจุไฟล์ INF และ device driver ไว้ด้วย

เมื่อทำการ enumeration เสร็จสมบูรณ์แล้ว windows จะเพิ่ม device ตัวใหม่ไปไว้ใน Device Managerที่แสดงอยู่ใน Control Panal ดังรูปที่ 2.14



รูปที่ **2.14** Device Managerใน Windows แสดงdeviceที่ตรวจพบ

ขั้นตอนการenumeration

1. ให้ผู้ใช้ทำการเสียบอุปกรณ์เข้าที่พอร์ต USB
2. hub นั้นจะทำการตรวจหา device
3. host เรียนรู้ device ที่เข้ามาใหม่
4. hub จะ reset ตัว device
5. hub สร้างเส้นสัญญาณระหว่าง device และ bus
6. hub ตรวจหาความเร็วของ device
7. host ส่ง requestไปหา maximum packet size ของ pipe
8. host ทำการกำหนดค่าaddress
9. host เรียนรู้เกี่ยวกับความสามารถของdevice
10. host จะ assigns และ loads ตัวdevice driver
11. device driver ทำการเลือก configuration

2.4 การ Install Device Driver

ในการ Install Device Driver จะใช้ไฟล์ที่มีนามสกุล .inf เราเรียกไฟล์ชนิดนี้ว่าไฟล์ INF เราจะใช้ไฟล์ชนิดนี้ในการ install device driver ซึ่งรายละเอียดภายในไฟล์นี้จะมีการระบุข้อมูลต่างๆของอุปกรณ์ฮาร์ดแวร์ และมีรหัสของอุปกรณ์ด้วย รายละเอียดมีดังนี้

```
[Version]
..
Class=USB
..
[DestinationDirs]
DefaultDestDir=10,System32\Drivers

[SourceDisksFiles]
test.sys=1

[SourceDisksNames]
1=%INSTDISK%,,,objchk\i386

[DeviceList]
%DESCRIPTION%=DriverInstall,
USB\VID_vvvv&PID_dddd&REV_rrrr[DriverInstall.NT]
CopyFiles=DriverCopyFiles

[DriverCopyFiles]
test.sys,,,
test.jnk,,,

[Strings]
INSTDISK="Project Disc"
DESCRIPTION="Test Project"
```

จากตัวอย่างไฟล์ INF ข้างบนมีการกำหนด class จะเป็นการแบ่งประเภทของอุปกรณ์ฮาร์ดแวร์ ซึ่งมีอยู่หลายประเภทดังแสดงในตาราง

ตารางที่ 2.1 Device classes ที่ใช้ภายในไฟล์ INF

ชื่อ Class	คำอธิบาย
1394	IEEE 1394 host bus controllers (but not peripherals)
Battery	Battery devices
CDROM	อุปกรณ์จำพวก CD ROM
DiskDrive	อุปกรณ์ฮาร์ดดิสก์
Display	การ์ดแสดงผล
FDC	Floppy disk controllers
FloppyDisk	อุปกรณ์ฟล็อปปี้ดิสก์
HDC	Hard disk controllers
HIDClass	Human input devices
Image	Still-image capture devices, including cameras and scanners
Infrared	NDIS miniport drivers for Serial-IR and Fast-IR ports
Keyboard	อุปกรณ์คีย์บอร์ด
MediumChanger	SCSI media changer devices
Media	Multimedia devices, including audio, DVD, joysticks, and full-motion video capture devices
Modem	โมเดม
Monitor	มอนิเตอร์
Mouse	อุปกรณ์เมาส์
MTD	Memory technology driver for memory devices
Multifunction	อุปกรณ์หลายฟังก์ชันในเครื่องเดียว
MultiportSerial	Intelligent multiport serial cards
Net	การ์ดแลน
NetClient	Network file system and print providers (client side)
NetService	Server-side support for network file systems
NetTrans	Network protocol drivers

ตารางที่ 2.1 Device classes ที่ใช้ภายในไฟล์ INF (ต่อ)

ชื่อ Class	คำอธิบาย
PCMCIA	PCMCIA and CardBus host controllers (but not peripherals)
Ports	อุปกรณ์ พอร์ตขนานและพอร์ตอนุกรม
Printer	ปริ้นเตอร์
SCSIAdapter	SCSI and RAID controllers, host bus adapter miniports, and disk array controllers
SmartCardReader	Smart card readers
System	System devices
TapeDrive	เทปไดร์ฟ
USB	USB host controllers and hubs (but not peripherals)
Volume	Logical storage volume drivers

ถัดจากการกำหนด Class ที่อยู่ในส่วนของ **[Version]** ก็จะเป็นการกำหนดค่าในส่วนของ **[DestinationDirs]** เป็นการกำหนดว่าจะให้ทำการ install device driver แล้วนำไฟล์ .sys ที่เป็นไฟล์ driver ไปเก็บไว้ที่พาธไหน ในที่นี้ให้นำไปเก็บไว้ที่ พาธ C:\WINDOWS\system32\drivers กรณีที่มีการลงระบบปฏิบัติการ windows ไว้ที่ ไดร์ฟ C แต่เราไม่สามารถรู้ได้ว่าผู้ใช้ได้ลงระบบปฏิบัติการ widows ไว้ที่ พาธไหน เราจึงใช้ตัวแปรระบบ 10 ซึ่งจะเป็นการอ้างอิงถึงไดร์ฟที่ผู้ใช้ลงระบบปฏิบัติการ windows ไว้ แล้วไปที่ไดเรกทอรี่ WINDOWS

จากนั้นจะเป็นส่วนของ **[SourceDisksFiles]** เป็นส่วนที่ระบุว่า ไฟล์ไดร์ฟเวอร์ชื่ออะไร ในส่วนของ **[SourceDisksNames]** เป็นการระบุว่าไฟล์ไดร์ฟเวอร์เก็บไว้ในไดเรกทอรี่อะไร ในดิสก์ชื่อว่าอะไร ค่า %INSTDISK% เป็นตัวแปรที่บอกถึงชื่อดิสก์ เนื่องจากการ install device driver จะทำโดยผ่านทางดิสก์ที่มาจากผู้ผลิตอุปกรณ์ฮาร์ดแวร์ และค่า %INSTDISK% จะมีการกำหนดในอยู่ในส่วนท้ายของไฟล์ INF จะอยู่ในส่วนของ String ส่วนของ **[DeviceList]** เป็นการบอกถึงรายละเอียดของอุปกรณ์ฮาร์ดแวร์ ซึ่งจะมีการกำหนดรหัสของอุปกรณ์ฮาร์ดแวร์ที่มาจากบริษัทผู้ผลิตด้วย ซึ่งจะอธิบายอยู่ในส่วนของ Device Identifiers ส่วน **[DriverCopyFiles]** จะเป็นการกำหนดว่าให้ทำการคัดลอกไฟล์ใดบ้างไปไว้ที่พาธ C:\WINDOWS\system32\drivers กรณีที่มีการลงระบบปฏิบัติการ windows ไว้ที่ ไดร์ฟ C ส่วน **[Strings]** เป็นการกำหนดค่าตัวแปร string ที่ใช้ภายในไฟล์ INF นี้

2.4.1 Device Identifiers

ในอุปกรณ์ฮาร์ดแวร์ทุกชนิดที่มีการทำงานเป็นแบบ Plug and Play คือ อุปกรณ์ที่นำมาเสียบต่อเข้ากับเครื่องคอมพิวเตอร์แล้วก็ติดตั้ง device driver ก็สามารถใช้ได้เลย อุปกรณ์ฮาร์ดแวร์ที่มีการทำงานเป็นแบบ Plug and Play นี้จะมีรหัสของอุปกรณ์ซึ่งจะมีค่าไม่ซ้ำกันโดยการกำหนดค่ารหัสของอุปกรณ์นี้ในแต่ละชนิดของอุปกรณ์ก็จะมีการกำหนดค่าที่ไม่เหมือนกัน โดยจะมีการกำหนดค่าและแบ่งชนิดดังนี้

2.4.1.1 อุปกรณ์ PCI

มีรูปแบบการกำหนดดังนี้

```
PCI\VEN_vvvv&DEV_dddd&SUBSYS_ssssssss&REV_rr
```

vvvv คือ รหัสผู้ผลิต

dddd คือ รหัสอุปกรณ์

ssssssss คือ รหัสย่อยของอุปกรณ์ ส่วนมากจะเป็น 0

rr คือ รหัสการแก้ไขปรับปรุง

2.4.1.2 อุปกรณ์ PCMCIA

มีรูปแบบการกำหนดดังนี้

```
PCMCIA\Manufacturer-Product-Crc
```

Manufacture คือ ชื่อผู้ผลิต

Product คือ ชื่ออุปกรณ์

Crc คือ เลขฐาน 16 จำนวนวน 4 บิต เป็นค่า CRC checksum ของอุปกรณ์

2.4.1.3 อุปกรณ์ SCSI

มีรูปแบบการกำหนดดังนี้

```
SCSI\tttvvvvvvvvpppppppppppppppprrrr
```

tttt คือ โค้ดชนิดของอุปกรณ์

vvvvvvvv คือ ชื่อผู้ผลิต โดยแสดงเป็นตัวอักษร 8 ตัว

ppppppppppppppppppppppp คือ ชื่อรหัสของอุปกรณ์ โดยแสดงเป็นตัวอักษร 16 ตัว

rrrr คือ รหัสการแก้ไขปรับปรุง

2.4.1.4 อุปกรณ์ IDE

มีรูปแบบการกำหนดดังนี้

```
IDE\ttttvpvprrrrrrrr
IDE\vpvprrrrrrrr
IDE\ttttvpvp
vpvprrrrrrrr
gggg
```

tttt คือ โค้ดชนิดของอุปกรณ์

vpvp คือ ชื่อผู้ผลิตแล้วตามด้วยชื่อของอุปกรณ์ โดยคั่นด้วยเครื่องหมาย underscore (_)

rrrrrrrr คือ รหัสการแก้ไขปรับปรุง โดยแสดงเป็นตัวอักษร 8 ตัว

gggg คือ โค้ด generic ที่แสดงชนิดของอุปกรณ์

2.4.1.5 อุปกรณ์ USB

มีรูปแบบการกำหนดดังนี้

```
USB\VID_vvvv&PID_dddd&REV_rrrr
```

vvvv คือ เลขฐาน 16 จำวนวน 4 บิต เป็นรหัสผู้ผลิต

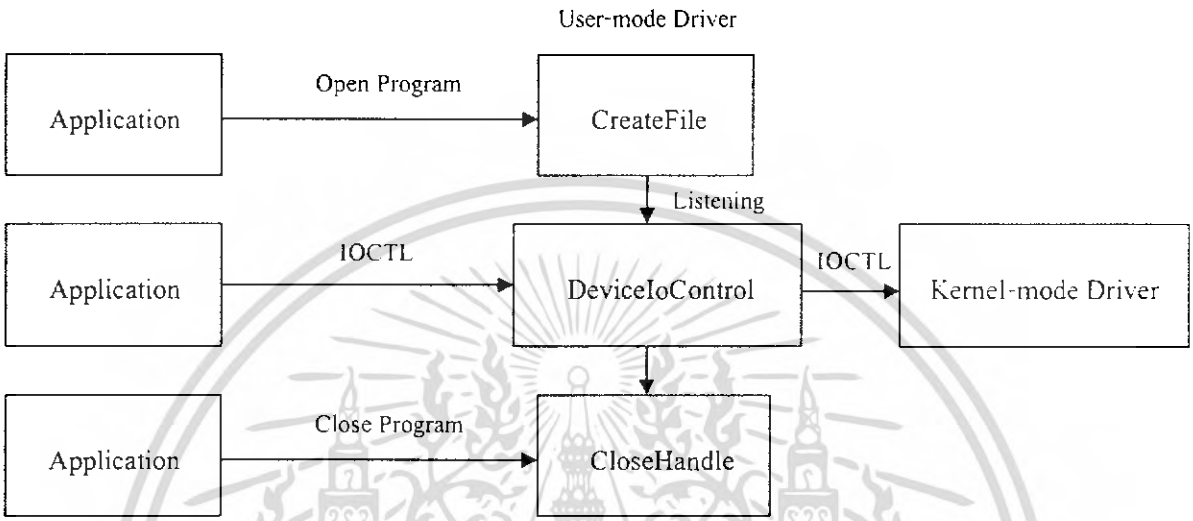
dddd คือ เลขฐาน 16 จำวนวน 4 บิต เป็นรหัสของอุปกรณ์

rrrr คือ รหัสการแก้ไขปรับปรุง

บทที่ 3

การออกแบบและการพัฒนา

3.1 การออกแบบ User-mode Driver



รูปที่ 3.1 แสดงการทำงานระหว่าง Application กับ User-mode Driver

การทำงานของ user-mode driver จะทำงานโดยคอยตอบสนองต่อการทำงานของ Application โดยที่ user-mode diver ส่วนใหญ่จะอยู่ในรูปของไฟล์ DLL มีการทำงานหลักๆ อยู่ 3 ขั้นตอน ดังนี้

1. Create File
2. ทำการรอ เพื่อตอบสนองการทำงานของ Application
3. Close File

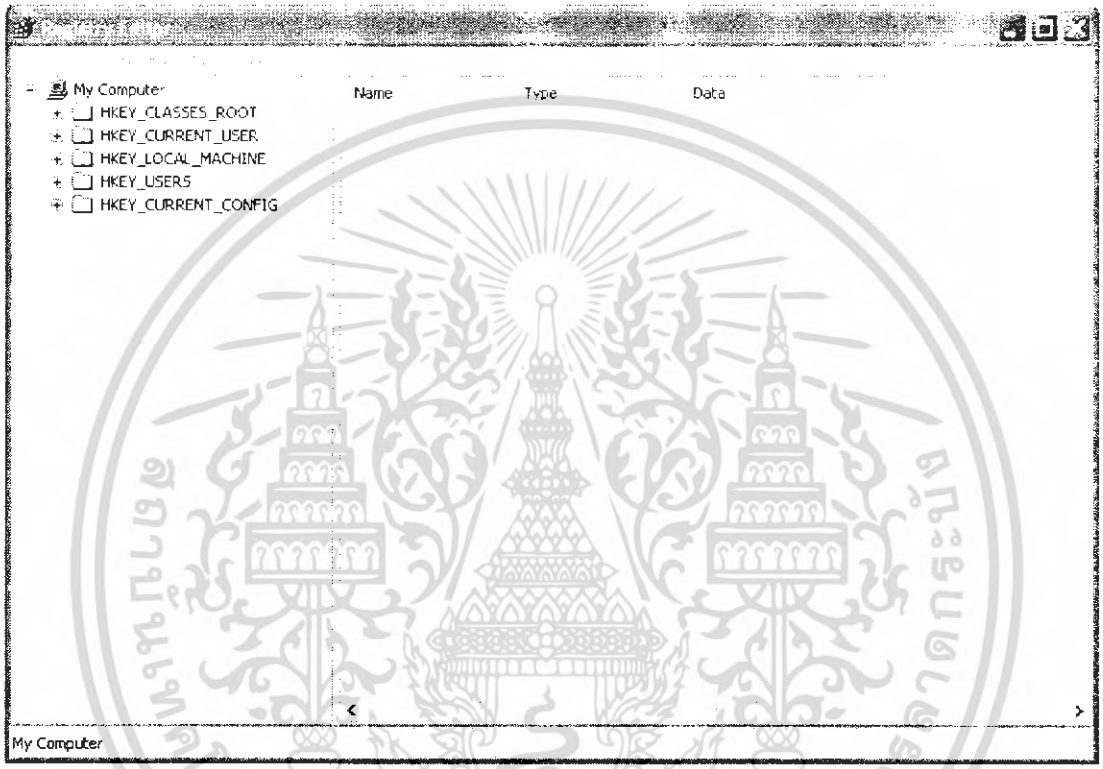
3.1.1 Create File

จะเป็นการสร้างไฟล์ Handle ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อโดยมีรูปแบบฟังก์ชันดังนี้

```
CreateFile("\\\\.\\Ezusb-0", ..., 0, NULL);
```

ค่าพารามิเตอร์ที่ส่งเข้าไปในฟังก์ชันคือ Device name ในที่นี้คือ Ezusb-0 และพาธของการ Create File คือ \\.\ จะเป็ นรูทของระบบเช่น C:\windows\system32 ระบบปฏิบัติการจะทำการแปลงเป็น \device\ HarddiskVolume1\Partition0 แต่เราไม่สามารถที่รู้ได้ว่าผู้ใช้ทั่วไปลง

ระบบปฏิบัติไว้ที่ใด จึงค่าเป็นรูทของระบบซึ่งจะมี Symboliclink ที่จะทำการเชื่อมโยงไปยัง Device name ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อได้ หากเราจะต้องการตรวจสอบว่า Device name ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อของเราได้ใส่เข้าไปในระบบหรือยัง ก็สามารถทำการตรวจสอบได้โดยการเข้าไปดูที่ Registry ของระบบปฏิบัติการ Windows เข้าไปที่ Run... พิมพ์คำว่า regedit กด OK จากนั้นเราก็จะเข้าสู่หน้าต่าง Registry Editor ของระบบปฏิบัติการ Windows ดังรูปที่ 3.2

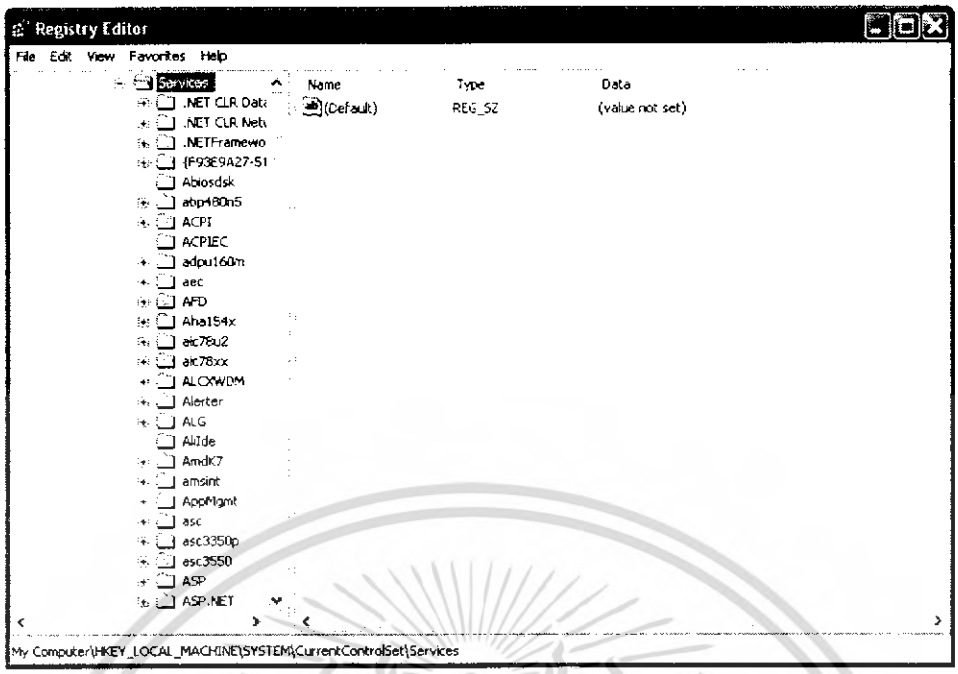


รูปที่ 3.2 หน้าต่าง Registry Editor ของระบบปฏิบัติการ Windows

หากเราจะต้องการตรวจสอบว่า Device name ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อของเราได้ใส่เข้าไปในระบบหรือยัง ให้เข้าไปดูที่พาธ

RegistryPath = HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

ดังรูปที่ 3.3 จะแสดง Device name ของอุปกรณ์ฮาร์ดแวร์ ที่มีอยู่ภายในเครื่อง ที่พาธ HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services ซึ่งจะมี Device name ของอุปกรณ์ฮาร์ดแวร์หลายชนิด ที่ได้ทำการติดตั้ง Device Driver ลงไปในเครื่องระบบปฏิบัติการ Windows



รูปที่ 3.3 พาธ HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

โดยรูปแบบของโปรแกรมที่เขียนขึ้นมามีดังนี้

```
char completeDeviceName[64] = "";
char pcMsg[64] = "";
strcat (completeDeviceName,"\\\\.\\" );
strcat (completeDeviceName, " Ezusb-0");
*phDeviceHandle = CreateFile(   completeDeviceName,
                                GENERIC_WRITE,
                                FILE_SHARE_WRITE,
                                NULL,
                                OPEN_EXISTING, 0,
                                NULL);
if (*phDeviceHandle == INVALID_HANDLE_VALUE) {
        return (FALSE);
} else {
        return (TRUE);
}
```


สิ่งที่ได้จากการ Create File คือ Handle ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อด้วย

3.1.2 ทำการรอ เพื่อตอบสนองการทำงานของ Application

user-mode driver เมื่อทำการ Create File และได้ Handle ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อเรียบร้อยแล้ว ก็จะทำการรอคอยเพื่อตอบสนองการทำงานต่อ Application เมื่อ Application ต้องการที่จะเขียนข้อมูลไปยังอุปกรณ์ฮาร์ดแวร์ ก็จะส่ง IoControlCode ไปให้กับ user-mode driver และ user-mode driver ก็จะทำการส่งข้อมูลที่จะเขียนไปยัง kernel-mode driver

ในการส่งข้อมูล user-mode driver จะส่งข้อมูลไปเป็นแพ็คเก็จ โดย 1 แพ็กเก็จจะส่งคำสั่งไปจำนวน 3 ชุด ฟังก์ชันที่ใช้ในการส่งข้อมูลมีรูปแบบดังนี้

DeviceIoControl(Handle, Code, InputData, InputLength,
OutputData, OutputLength);

Handle	Handle ของอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อ
Code	IoControlCode
InputData	ข้อมูลที่ต้องการจะเขียน
InputLength	ขนาดข้อข้อมูลที่จะเขียน
OutputData	ข้อมูลที่ถูกส่งกลับมา
OutputLength	ขนาดข้อข้อมูลที่ส่งกลับมา

ค่าที่ได้รับกลับมาจากฟังก์ชันนี้คือ BOOL

ถ้าเป็น TRUE แสดงว่าการส่งข้อมูลเป็นผลสำเร็จ

ถ้าเป็น FALSE แสดงว่าการส่งข้อมูลเป็นไม่ผลสำเร็จ

ตัวอย่าง IOCTL (I/O Control Code) ที่ส่งออกไปให้กับ kernel-mode driver

- *IOCTL_EZUSB_BULK_WRITE*
- *IOCTL_EZUSB_BULK_READ*

สำหรับการเขียนข้อมูล 1 ครั้ง จะส่งข้อมูลไป 3 ชุด

ชุดที่1 จะเป็นการส่ง command ออกไปให้กับอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อโดยจะส่งข้อมูลออกไปทาง PIPE 6 เพื่อเป็นการเซ็ตค่าให้กับอุปกรณ์ฮาร์ดแวร์

ชุดที่2 จะเป็นการส่งข้อมูล ออกไปให้กับอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อโดยจะส่งข้อมูลออกไปทาง PIPE 4 เพื่อเป็นการบอกว่าจะเขียนข้อมูลออกไปที่ address ที่เท่าไหร่

ชุดที่3 จะเป็นการส่งข้อมูล ออกไปให้กับอุปกรณ์ฮาร์ดแวร์ที่ต้องการจะติดต่อโดยจะส่งข้อมูลออกไปทาง PIPE 4 เพื่อเป็นการบอกว่าข้อมูลที่ต้องการจะเขียนคืออะไร

โดยรูปแบบของโปรแกรมที่เขียนขึ้นมีดังนี้

```
bulkControl.pipeNum = 6;
if (bOpenDriver (&hDevice, pcDriverName) != TRUE){
        hDevice = NULL;
}
if (hDevice == NULL){
        m_output += "PC didn't connect board\x0d\x0a";
}
if (hDevice != NULL){
        bResult = DeviceIoControl(hDevice,
                                ioctl_val,
                                &bulkControl,
                                sizeof (BULK_TRANSFER_CONTROL),
                                command,
                                5,
                                (unsigned long *)&nBytes,
                                NULL);
        bulkControl.pipeNum = 4;
        bResult = DeviceIoControl(hDevice,
                                ioctl_val,
                                &bulkControl,
                                sizeof (BULK_TRANSFER_CONTROL),
                                command,
                                5,
                                (unsigned long *)&nBytes,
                                NULL);
        bulkControl.pipeNum = 2;
        bResult = DeviceIoControl(hDevice,
                                ioctl_val,
                                &bulkControl,
                                sizeof (BULK_TRANSFER_CONTROL),
                                &data,
```

```
length,
(unsigned long *)&nBytes,
NULL);
```

จากโปรแกรมที่เขียนขึ้นมามีการใช้ฟังก์ชัน DeviceIoControl ส่งข้อมูลออกไปเป็นจำนวน 3 ครั้ง โดยครั้งแรกเป็นการส่งออกไปที่ไปป์หมายเลข 6 เพื่อส่ง command ออกไป และค่าที่อยู่ในตัวแปร command คือค่าคำสั่งและค่า address เริ่มต้นที่ต้องการจะส่งข้อมูลออกไป โดยการส่งครั้งแรกออกไปที่ไปป์หมายเลข 6 เพื่อเป็นการตั้งค่าให้กับอุปกรณ์ฮาร์ดแวร์ว่าต้องการทำงานที่โหมดใด ถ้าไม่มีการตั้งค่านี้การส่งข้อมูลออกไป จะเป็นการเขียนข้อมูลออกไปที่ address เดิมบวกกับจำนวนขนาดของข้อมูล ดังนั้นค่า address จะบวกเพิ่มขึ้นทุกๆ ครั้งที่มีการส่งข้อมูล ส่วนการส่งครั้งที่ 2 ส่งออกไปที่ไปป์หมายเลข 4 เป็นการตั้งค่า address เริ่มต้นที่จะส่งข้อมูลออกไป ส่วนการส่งครั้งสุดท้าย จึงจะเป็นข้อมูลจริงที่ต้องการจะส่งออกไป จากโค้ดข้างบนจะมีส่วนตรวจเช็คว่าอุปกรณ์ได้ต่ออยู่กับเครื่องหรือไม่ ถ้าไม่ได้ต่ออยู่จะทำให้ไม่สามารถสร้างไฟล์ Handle ได้ ดังนั้นถ้าอุปกรณ์ไม่ได้ต่ออยู่ก็จะให้แสดงข้อความว่า "PC didn't connect board"

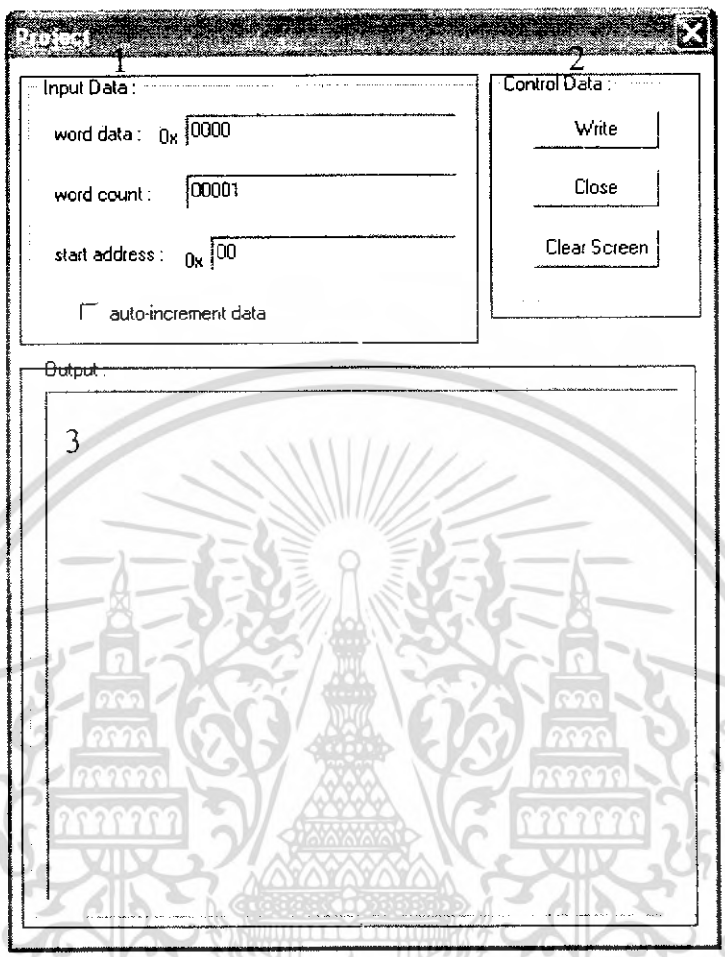
3.1.3 Close File

จะเป็นการปิดการเชื่อมต่อกับอุปกรณ์ฮาร์ดแวร์โดยมีรูปแบบฟังก์ชันดังนี้

```
CloseHandle(Handle);
```

ค่าพารามิเตอร์ที่ส่งเข้าไปในฟังก์ชันคือ Handle ของอุปกรณ์ฮาร์ดแวร์ที่ได้ทำการ Create File ได้ Create File เอาไว้ในขั้นตอนที่1

3.2 การออกแบบโปรแกรมที่ทำงานติดต่อกับ User-mode Driver



รูปที่ 3.4 User interface ของโปรแกรม

1. ส่วนที่ 1

1. Edit box "word data": ใส่ข้อมูลที่ต้องการจะเขียนไปยัง board
2. Edit box "word count": ใส่จำนวนครั้งที่ต้องการจะส่งข้อมูลออกไป
3. Edit box "start address": ใส่ address เริ่มต้นของการส่งข้อมูล
4. Radio box "auto-increment address": เลือกถ้าผู้ใช้ต้องการให้เพิ่ม address อัตโนมัติเมื่อทำการอ่านเขียน

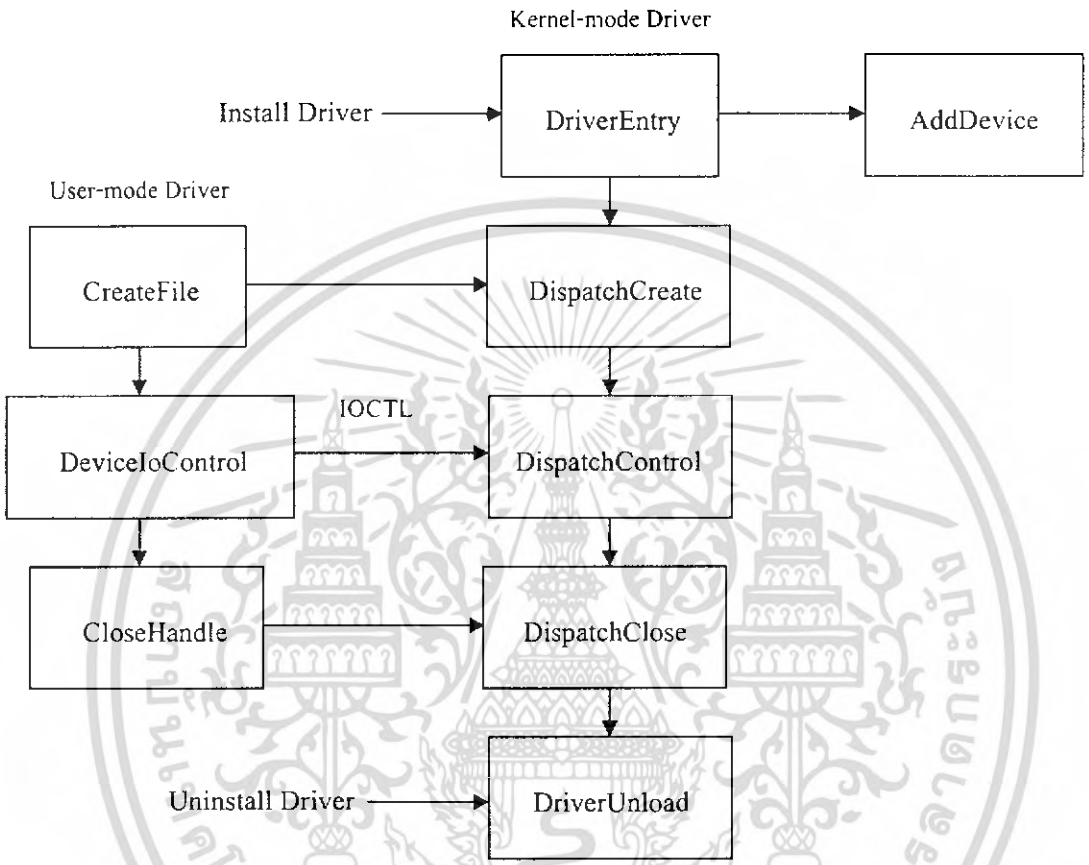
2. ส่วนที่ 2

1. Button "Write": คลิกเพื่อเขียนข้อมูลไปยัง board
2. Button "Close": คลิกเพื่อปิดโปรแกรม
3. Button "Clear Screen": คลิกเพื่อลบข้อมูลทั้งหมดใน edit box

3. ส่วนที่ 3

1. Edit box ที่ใหญ่ที่สุด: รายงานข้อมูลที่ถูกเขียนและ address ที่ถูกเขียน

3.3 การออกแบบ Kernel-mode Driver



รูปที่ 3.5 แสดงการทำงานระหว่าง User-mode Driver กับ kernel-mode Driver

การทำงานของ kernel-mode driver จะทำงานโดยคอยตอบสนองต่อการทำงานของ user-mode diver โดยที่ kernel-mode diver จะอยู่ในรูปของไฟล์ SYSTEM (*.sys) มีการทำงานของฟังก์ชันหลักๆ อยู่ 6 ฟังก์ชัน ดังนี้

1. DriverEntry
2. AddDevice
3. DispatchCreate
4. DispatchControl
5. DispatchClose
6. DriverUnload

3.3.1 DriverEntry

เป็นฟังก์ชันของ kernel-mode driver เพราะเป็นจุดทางเข้าไปใน kernel-mode driver การทำงานส่วนใหญ่จะเป็นการกำหนดค่าเริ่มต้นให้กับ driver คือเมื่อมี IRP อะไรเข้ามาจะให้ไปทำงานที่ฟังก์ชันไหน มีรูปแบบของฟังก์ชันดังนี้

```
DriverEntry( IN PDRIVER_OBJECT DriverObject,IN PUNICODE_STRING
RegistryPath)
```

ค่าพารามิเตอร์ตัวแรกที่เข้ามา คือ DriverObject เป็นตัวแปร Struct ที่มี data member เป็น pointer to function เป็นการกำหนดค่าเริ่มต้นเพื่อบอกให้ driver รู้ว่าให้ไปทำงานต่อที่ฟังชันอะไร เช่น

```
DriverObject->DriverUnload = DriverUnload;
DriverObject->DriverExtension->AddDevice = AddDevice;
DriverObject->MajorFunction[IRP_MJ_CREATE] = DispatchCreate;
DriverObject->MajorFunction[IRP_MJ_CLOSE] = DispatchClose;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = DispatchControl;
```

ค่าพารามิเตอร์ตัวที่2 คือ RegistryPath เป็นตัวแปรที่เก็บค่าของ registry path ที่ driver จะทำการใส่ device name ลงไปในระบบ ซึ่งค่า RegistryPath จะมีค่าดังนี้

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
```

3.3.2 AddDevice

เป็นฟังก์ชันที่ใช้ในการกำหนดค่าเริ่มต้นและทำการสร้าง DeviceObject ซึ่งเป็นตัวแปรที่เก็บค่า IRP และค่าต่างๆ ของ driver มีรูปแบบของฟังก์ชันดังนี้

```
AddDevice(PDRIVER_OBJECT DriverObject, PDEVICE_OBJECT deviceObject )
```

ค่าพารามิเตอร์ตัวแรกที่เข้ามา คือ DriverObject เป็นตัวแปร Struct ตัวเดียวกันกับที่อยู่ใน DriverEntry ที่มี data member เป็น pointer to function เป็นการกำหนดค่าเริ่มต้นเพื่อบอกให้ driver รู้ว่าให้ไปทำงานต่อที่ฟังชันอะไร

ค่าพารามิเตอร์ตัวที่2 คือ DeviceObject เป็นตัวแปร Struct ที่มี data member เป็นค่าการทำงานต่างๆ ที่ได้รับเข้ามาจากระบบปฏิบัติการและเป็นค่าที่ได้รับเข้ามาจาก user-mode driver การทำงานหลักๆ ของฟังก์ชันนี้คือ

1. ทำการสร้าง device name โดยใช้ฟังก์ชัน IoCreateDevice มีรูปแบบดังนี้

```
NTSTATUS  IoCreateDevice (DriverObject, sizeof (DEVICE_EXTENSION),
                          &deviceNameUnicodeString,
                          FILE_DEVICE_UNKNOWN, 0, FALSE.
                          &deviceObject);
```

2. ทำการสร้าง Symboliclink โดยการใช้ฟังก์ชัน IoCreateSymboliclink มีรูปแบบดังนี้

```
NTSTATUS  IoCreateSymbolicLink (&deviceLinkUnicodeString,
                                &deviceNameUnicodeString);
```

3.3.3 DispatchCreate

เป็นฟังก์ชันที่คอบสนองต่อการทำงานของ user-mode driver เมื่อมีการใช้ฟังก์ชัน CreateFile ที่โปรแกรมบน user-mode driver ก็จะมีการทำงานบน kernel-mode driver ที่ฟังก์ชัน DispatchCreate มีรูปแบบของฟังก์ชันดังนี้

```
NTSTATUS DispatchCreate(PDEVICE_OBJECT deviceObject, PIRP Irp)
```

การทำงานของฟังก์ชันนี้จะมีการเรียกใช้ ฟังก์ชัน IoGetCurrentIrpStackLocation(Irp) เพื่อนำค่าของ stack ออกมา เพื่อนำมาหา IRP ที่ได้รับเข้ามาเพื่อตอบสนองต่อการทำงาน และจะมีการเรียกใช้ฟังก์ชัน InterlockedIncrement(&pdx->handles) เพื่อทำการล็อกการทำงานเพื่อใช้ในการทำงานแบบ Syncronization เพราะการทำงานของ driver จะทำงานที่ละ request เพราะฉะนั้นจึงต้องมีการล็อก เพื่อไม่ให้ request อื่นมาแทรกการทำงาน

3.3.4 DispatchControl

เป็นฟังก์ชันที่ตอบสนองต่อการทำงานของ user-mode driver เมื่อมีการใช้ฟังก์ชัน DeviceIoControl ที่โปรแกรมบน user-mode driver ก็จะมีการทำงานบน kernel-mode driver ที่ฟังก์ชัน DispatchControl มีรูปแบบของฟังก์ชันดังนี้

```
NTSTATUS DispatchControl(PDEVICE_OBJECT fdo, PIRP Irp)
```

การทำงานก็จะทำงานตอบสนองต่อ IOCTL ที่ได้รับเข้ามาจาก user-mode driver และนำมาเปรียบเทียบ IOCTL และทำงานตาม IOCTL ที่ได้รับเข้ามา การทำงานจะมีดังนี้

```
PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(Irp);
ULONG cbin = stack->Parameters.DeviceIoControl.InputBufferLength;
ULONG cbout = stack->Parameters.DeviceIoControl.OutputBufferLength
ULONG code = stack->Parameters.DeviceIoControl.IoControlCode;

switch (code){
        case IOCTL_EZUSB_BULK_WRITE :
...
}
```

เมื่อได้รับ IOCTL เข้ามาจาก user-mode driver ก็จะนำมาหาค่าของ InputBuffer จากคำสั่ง stack->Parameters.DeviceIoControl.InputBufferLength และหาค่าของ OutputBuffer จากคำสั่ง stack->Parameters.DeviceIoControl.OutputBufferLength และหาค่าของ IOCTL จากคำสั่ง stack->Parameters.DeviceIoControl.IoControlCode เมื่อได้ IOCTL แล้วก็จะนำมาเปรียบเทียบเพื่อทำงานต่อไป

3.3.5 DispatchClose

เป็นฟังก์ชันที่ตอบสนองต่อการทำงานของ user-mode driver เมื่อมีการใช้ฟังก์ชัน CloseFile ที่โปรแกรมบน user-mode driver ก็จะมีการทำงานบน kernel-mode driver ที่ฟังก์ชัน DispatchCreate มีรูปแบบของฟังก์ชันดังนี้

```
NTSTATUS DispatchClose(PDEVICE_OBJECT deviceObject, PIRP Irp)
```

การทำงานของฟังก์ชันนี้จะมีการเรียกใช้ ฟังก์ชัน IoGetCurrentIrpStackLocation(Irp) เพื่อนำค่าของ stack ออกมา เพื่อนำมาหา IRP ที่ได้รับเข้ามาเพื่อตอบสนองต่อการทำงาน และจะมีการเรียกใช้ฟังก์ชัน InterlockedDecrement(&pdx->handles) เพื่อทำการปลดล็อกการทำงานเพื่อที่ใช้ในการทำงานแบบ Syncronization เพราะเมื่อมีการทำงานเสร็จแล้วจะต้องทำการปลดล็อกเพื่อให้ request อื่นเข้ามาทำงานต่อไป

3.3.6 DriverUnload

จะทำงานก็ต่อเมื่อมีการ Uninstall driver การทำงานจะเป็น Deallocate ตัวแปรต่างๆ ที่ได้ทำการสร้างขึ้นมาใช้ มีรูปแบบดังนี้

VOID DriverUnload(IN PDRIVER_OBJECT DriverObject)

ในการเขียน driver ทั่วๆไป จะมีการใช้ตัวแปรชนิด UnicodeString ซึ่งจะต้องมีการสร้างตัวแปรชนิดนี้ขึ้นมา เช่น ใช้ในการกำหนด device name ก็จะใช้ตัวแปรแบบ UnicodeString เพราะฉะนั้น เมื่อทำการ Uninstall driver จะต้องใช้คำสั่ง RtlFreeUnicodeString(&servkey) ในการทำลายตัวแปรนี้

3.4 การออกแบบไฟล์ INF

เป็นไฟล์ที่ใช้ในการ install device driver ซึ่งรายละเอียดภายในไฟล์นี้จะมีการระบุข้อมูลต่างๆของอุปกรณ์ฮาร์ดแวร์ และมีรหัสของอุปกรณ์ด้วย โดยอุปกรณ์ที่ใช้เป็นอุปกรณ์ USB มีการกำหนดรายละเอียดมีดังนี้

```
[Version]
..
Class=USB
..
[DestinationDirs]
DefaultDestDir=10,System32\Drivers

[SourceDisksFiles]
test.sys=1

[SourceDisksNames]
1=%INSTDISK%,,,objchk\i386

[DeviceList]
%DESCRIPTION%=DriverInstall,
USB\VID_vvvv&PID_dddd&REV_rrrr[DriverInstall.NT]
CopyFiles=DriverCopyFiles

[DriverCopyFiles]
test.sys,,,
test.jnk,,,
```

```
[Strings]
INSTDISK="Project Disc"
DESCRIPTION="Test Project"
```

ส่วน **[Version]** กำหนด Class เป็น USB

ส่วน **[DestinationDirs]** กำหนดพาธไปที่ C:\WINDOWS\system32\drivers และใช้ตัวแปรระบบ 10 เพื่อในกรณีที่ผู้ใช้ไม่ได้ลงระบบปฏิบัติการ windows ไว้ที่ไดร์ว C กำหนดพาธเพื่อใช้ในกรณีที่มีการคัดลอกไฟล์ระบบของอุปกรณ์ฮาร์ดแวร์ จะได้ทำการคัดลอกไปไว้ที่พาธ C:\WINDOWS\system32\drivers เพื่อนำไปใช้กับ driver และระบบปฏิบัติการ windows

ส่วน **[SourceDisksFiles]** กำหนดว่า ไฟล์ไดร์วเวอร์ชื่อ test.sys

ส่วน **[SourceDisksNames]** กำหนดว่าไฟล์ไดร์วเวอร์อยู่พาธ objchk\i386 ของดิสก์

ส่วน **[DeviceList]** กำหนดรหัสของอุปกรณ์ และมีการกำหนดให้มีการคัดลอกไฟล์ไปไว้ที่พาธ C:\WINDOWS\system32\drivers ด้วย โดยในโครงงานนี้เป็นอุปกรณ์ USB มีการกำหนดค่าดังนี้

```
USB\VID_vvvv&PID_dddd&REV_rrrr
```

vvvv คือ เลขฐาน 16 จำนวนวน 4 บิต เป็นรหัสผู้ผลิต

dddd คือ เลขฐาน 16 จำนวนวน 4 บิต เป็นรหัสของอุปกรณ์

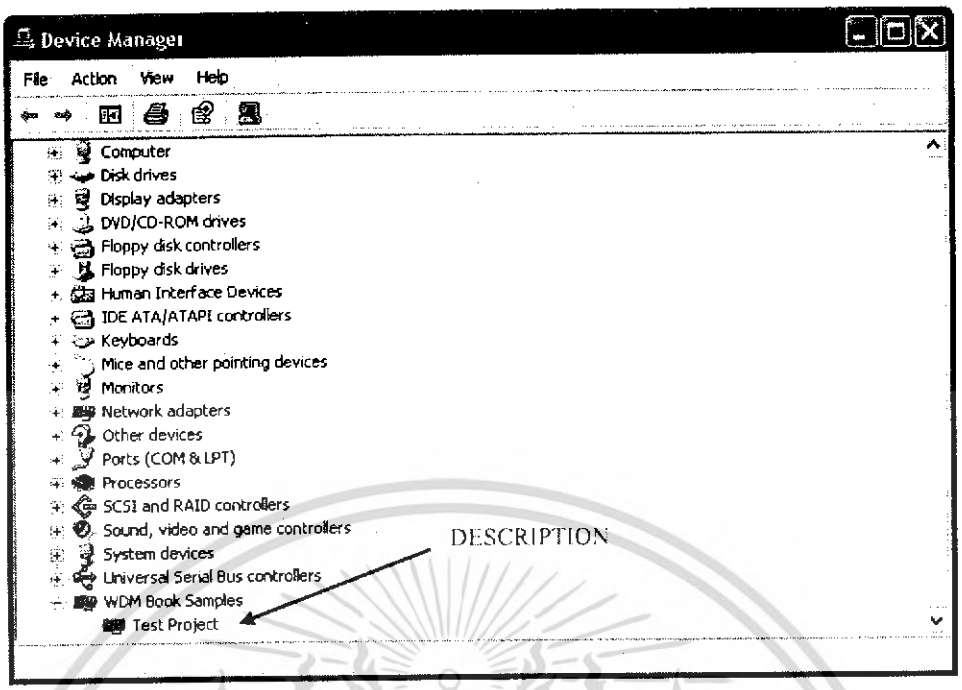
rrrr คือ รหัสการแก้ไขปรับปรุง

ส่วน **[DriverCopyFiles]** เป็นการกำหนดให้การ install device driver ให้ทำการคัดลอกไฟล์ 2 ไฟล์คือ test.sys และ test.jnk ซึ่ง test.sys เป็นไฟล์ไดร์วเวอร์ และ test.jnk เป็นไฟล์ที่ใช้ทำงานร่วมกับไดร์วเวอร์ ไปไว้ที่พาธ C:\WINDOWS\system32\drivers

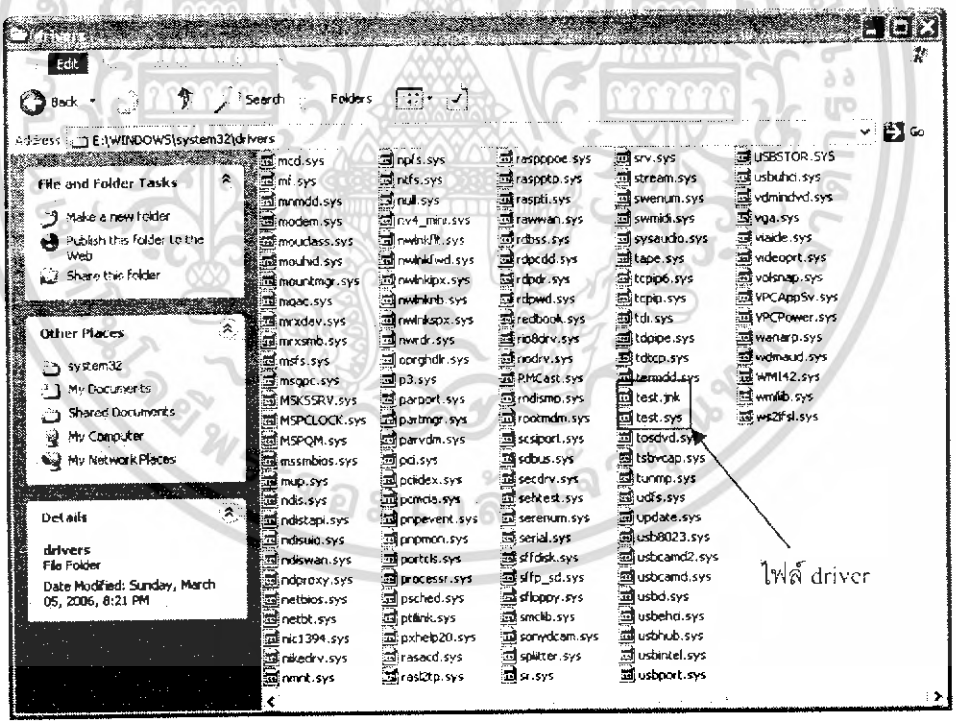
ส่วน **[Strings]** เป็นการกำหนดค่าตัวแปร string ที่ใช้ภายในไฟล์ INF โดยมีการกำหนดไว้ 2 ตัวแปรคือ INSTDISK และ DESCRIPTION ซึ่งแต่ละตัวแปรมีรายละเอียดดังนี้

INSTDISK คือ ชื่อของดิสก์ที่เก็บไฟล์ driver เอาไว้

DESCRIPTION คือ ชื่อที่ใช้แสดงในหน้าต่าง Device Manager เมื่อทำการ install device driver เสร็จแล้วดังแสดงในรูปที่ 3.6



รูปที่ 3.6 หน้าต่าง Device Manager



รูปที่ 3.7 C:\WINDOWS\system32\drivers

จากรูปที่ 3.7 แสดงพาธ C:\WINDOWS\system32\drivers ที่เก็บไฟล์ test.sys และ test.jnk ซึ่งเป็นไฟล์ไดร์วเวอร์และใช้ที่ใช้ทำงานร่วมกับไฟล์ไดร์วเวอร์

3.5 ขั้นตอนในการออกแบบและพัฒนา USB

ในโปรเจคทั่วๆไป ถ้าคุณต้องการที่จะสร้างโปรเจคขึ้นมานั้น เราต้องทำการแบ่งงานออกเป็นส่วนๆก่อน ที่จะทำการลงมือทำ ซึ่งในการเขียนโปรแกรมภายในตัวชิปนั้น คุณสามารถเริ่มได้โดยการเขียนโค้ดให้ระบบปฏิบัติการนั้นให้พอที่จะหาตัวอุปกรณ์นั้น พบและสามารถทำการระบุไปที่อุปกรณ์ที่เรานั้นได้เขียนขึ้นมา จากนั้นคุณค่อยมาทำการแลกเปลี่ยนข้อมูลอย่างง่ายๆ กับตัวโปรแกรมที่รันอยู่บนระบบปฏิบัติการ จากนั้นคุณสามารถที่จะมาเพิ่มส่วนที่เป็นรายละเอียดปลีกย่อยที่จะทำให้สามารถติดต่อกับโปรแกรมที่รันอยู่บนระบบปฏิบัติการได้ ซึ่งขั้นตอนในการพัฒนาโปรเจคนั้น ก็จะประกอบด้วย การ initial decisions ,enumerating และ exchanging data

3.5.1 Initial decisions

ก่อนที่คุณจะเริ่มพัฒนาโปรเจคได้นั้น คุณจำเป็นทำการต้องรวบรวมข้อมูลและต้องทำบางอย่างในการที่จะใช้ในการตัดสินใจกับตัวโปรเจคของเรา

1. อย่างแรกคุณต้องระบุรายละเอียดตัวอุปกรณ์ของคุณสำหรับในการติดต่อกับตัวUSB ของคุณ ว่าคุณต้องการที่จะรับส่งข้อมูลที่มีขนาดมากเท่าไรและความเร็วในการรับส่งข้อมูลที่ตัวอุปกรณ์ที่คุณทำขึ้นมานั้นต้องการในการทำงาน
2. นำเอาคำตอบในข้อที่ 1 มาทำการระบุความต้องการของตัวชิปของเรา ถ้าคุณมีชิปที่เอาไว้อ้างอิงอยู่แล้วก็ให้กำหนดมันลงไป
3. นำเอาความต้องการของคุณมาใช้ในการตัดสินใจว่า คอมพิวเตอร์ของคุณจะติดต่อในการส่งข้อมูลกับตัวอุปกรณ์คุณโดยการใช้ตัว device driverทั่วๆไป หรือจะเป็นตัว device driver ที่คุณทำขึ้นมา
4. เลือก controller chip ที่ตรงกับความต้องการของคุณ

3.5.2 Enumerating

ในส่วนนี้ให้เรานำเอาอะไรก็ตามที่คุณจำเป็นต้องเอาหรือรับจากระบบปฏิบัติการมาทำการระบุตัวอุปกรณ์ของคุณ

1. การเขียนโค้ดควบคุมชิปตามรายการในการระบุตัวอุปกรณ์ ซึ่งรายละเอียดจะเปลี่ยนแปลงตามชิป ซึ่งชิปนั้นต้องสามารถติดต่อกับdescriptors ซึ่งชิปนั้นต้องมีโปรแกรม หรือ อุปกรณ์ที่สามารถจำและตอบสนองความต้องการของโค้ดในโปรแกรมที่โอสต์ส่งมาเมื่อมันทำการระบุตัวอุปกรณ์ ซึ่งผู้ขายชิปทั่วไปจะจัดเตรียมโค้ดตัวอย่างซึ่งคุณสามารถใช้กับการปรับปรุงเล็กๆน้อยๆ
2. สร้างหรือเอา INF ไฟล์ซึ่งระบบปฏิบัติการสามารถทำการชี้ไปที่อุปกรณ์ของคุณเมื่อทำการระบุมัน ซึ่งINF file name คือชื่อที่อุปกรณ์เรานั้นใช้ในส่วนนี้ คุณสามารถใช้ driver

ทั่วๆไปที่สามารถสนับสนุนชิปของคุณซึ่งผู้ขายส่วนมากจะจัดเตรียมตัวอย่าง INF file และ driver ไว้ให้คุณ

3. ถ้าจำเป็นต้องออกแบบและสร้างวงจรที่จำติดต่อกับชิปไปที่โฮสต์ซึ่งส่วนใหญ่ในกรณีที่คุณทำการใส่ค่าเริ่มต้นในการใช้พัฒนาบอร์ด
4. โหลดโค้ดของคุณไปสู่อุปกรณ์ของคุณและเสียบอุปกรณ์ของคุณใส่ Host และระบบปฏิบัติการควรจะทำการระบุตัวอุปกรณ์ และเพิ่มมันลงใน control panel และชี้ไปยังตัวที่ถูกต้อง

3.5.3 Exchanging Data

ขั้นตอนเหล่านี้มีความสัมพันธ์กับfunction ที่จะกระทำกับอุปกรณ์

1. เพิ่มความสามารถที่ตัวอุปกรณ์โดยเพิ่มโค้ดที่ตัวโปรแกรมภายในชิปและส่วนต่างๆที่ติดต่อกับชิป
2. ถ้าคุณใช้custom driver คุณต้องเขียนdriver ในการติดต่อกับอุปกรณ์
3. ถ้าจำเป็นที่ต้องเขียนโปรแกรมในการติดต่อกับอุปกรณ์ ซึ่งถ้าคุณกำลังออกแบบ mouse ,keyboard หรือ อุปกรณ์ทั่วไป คุณสามารถที่จะติดต่อกับอุปกรณ์ได้โดยโปรแกรมใดๆ
4. ทำการdebug และ วนทำซ้ำถ้าจำเป็น
5. เมื่อโค้ดถูกdebug แล้วคุณก็พร้อมที่จะโหลดโปรแกรมลงในชิป และให้ทำการทดสอบตัวอุปกรณ์รอบสุดท้าย

แต่ก่อนที่คุณเริ่มในส่วนนี้ มันต้องใช้ประโยชน์ในส่วนที่คุณรู้ในเรื่องเกี่ยวกับการระบุอุปกรณ์ และการรับส่งข้อมูลกับอุปกรณ์ ดังนั้นคุณสามารถไปใช้ในการเลือกชิปและdriverของคุณได้

3.5.4 Tools

ในการพัฒนาอุปกรณ์เกี่ยวกับ USB นั้น คุณจำเป็นต้องมี tool ที่จะนำมาช่วยเราในการพัฒนาโปรแกรมที่เราใช้ดังนี้

1. assembler หรือ compiler ที่สามารถสร้าง firmware(โค้ดที่รันอยู่ในcontroller chip) ได้ ถ้าคุณใช้โค้ด assembly คุณจำเป็นต้องทำการแปลงโค้ดที่เป็นภาษาassemblyของคุณให้ไปเป็น machine code ให้ controller เข้าใจ แต่ถ้าคุณใช้ภาษา ซี หรือ ภาษาอื่น คุณจำเป็นหาตัวที่สามารถทำการแปลงโค้ดให้มาเป็น machine code ให้ได้เพื่อจะนำมาใช้ในการควบคุมของคุณ
2. โปรแกรมเมอร์จะเก็บโค้ดที่ทำเสร็จแล้วไว้ใน controller program memory

3. โฮสต์ อาจประกอบด้วย device driver หรือ mini-driver หรือ application code ในการสร้าง device driver คุณจำเป็นต้องใช้ Visual C++ ที่ซึ่งสามารถทำ WDM(Win32 Driver Model) ซึ่งนำมาใช้กับ USB ได้
4. โปรแกรมที่ใช้ในการดักจับดูเหตุการณ์ที่เกิดขึ้นภายในโปรแกรม หรือ เครื่องมือที่ใช้ในการ debug อื่นๆสำหรับการพัฒนาโปรแกรม

3.6 อุปกรณ์ฮาร์ดแวร์

อุปกรณ์ฮาร์ดแวร์ชิ้นนี้เป็นผลิตภัณฑ์ของบริษัท Design Gateway Company Limited เป็นระบบที่สามารถใช้เชื่อมต่อคอมพิวเตอร์กับฮาร์ดแวร์ที่พัฒนาขึ้นเอง ทำให้ฮาร์ดแวร์นี้สามารถติดต่อกับพอร์ต USB ได้โดยที่ผู้ใช้ไม่ต้องมีความรู้เกี่ยวกับพอร์ต USB ทำให้ผู้ใช้สามารถพัฒนาผลิตภัณฑ์ได้เร็วขึ้น นอกจากนั้น timing diagram ของอุปกรณ์ฮาร์ดแวร์ชิ้นนี้ ยังเหมือนกับของหน่วยประมวลผล Z80 อีกด้วย

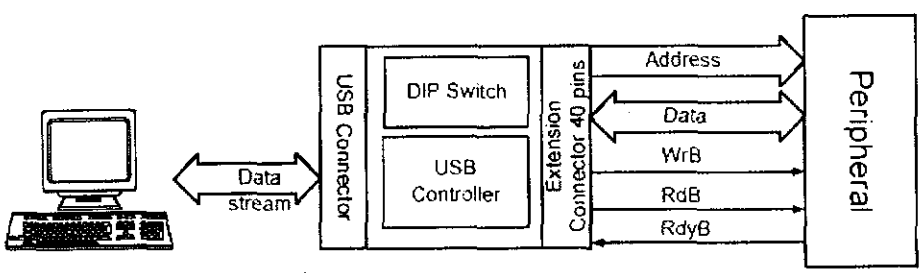
3.6.1 คุณลักษณะของอุปกรณ์ฮาร์ดแวร์

1. อุปกรณ์ฮาร์ดแวร์ชิ้นนี้เชื่อมต่อกับคอมพิวเตอร์ด้วยพอร์ต USB 1.1
2. อุปกรณ์ฮาร์ดแวร์ชิ้นนี้เชื่อมต่อกับฮาร์ดแวร์ที่ต้องการได้ด้วย connector 40 pin ซึ่งประกอบด้วย address bus 8 บิต, data bus 16 บิต, สัญญาณ read (RdB) 1 บิต, สัญญาณ write (WrB) 1 บิต, สัญญาณ ready (RdyB) 1 บิต, ground, และ Vcc

3.6.2 Block Diagram ของอุปกรณ์ฮาร์ดแวร์

Block diagram ของอุปกรณ์ฮาร์ดแวร์ชิ้นนี้ประกอบด้วย

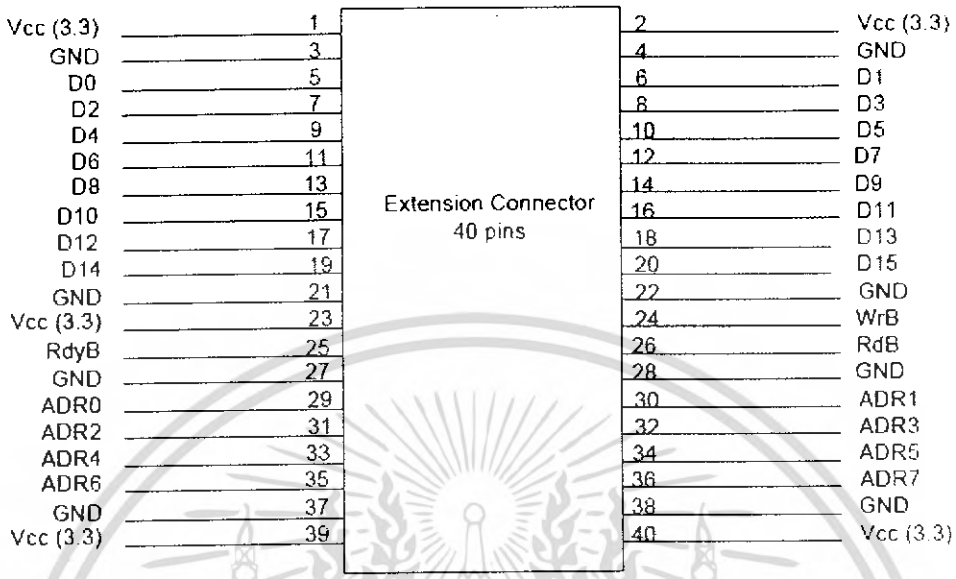
1. ชิพ USB controller ซึ่งทำหน้าที่ติดต่อระหว่างคอมพิวเตอร์และอุปกรณ์ที่ต้องการ
2. USB connector ถูกใช้เพื่อเสียบสาย USB ระหว่างคอมพิวเตอร์และวงจรของอุปกรณ์ฮาร์ดแวร์
3. DIP switch ถูกใช้เพื่อกำหนดหมายเลข board
4. Connector ขนาด 40 pin ถูกใช้เชื่อมต่อระหว่างอุปกรณ์ฮาร์ดแวร์ชิ้นนี้และอุปกรณ์ที่ต้องการ ประกอบด้วย data bus, address bus, write (WrB), read (RdB), และ ready (RdyB)



รูปที่ 3.8 Block diagram

3.6.3 ชื่อและตำแหน่ง

ขาสัญญาณ



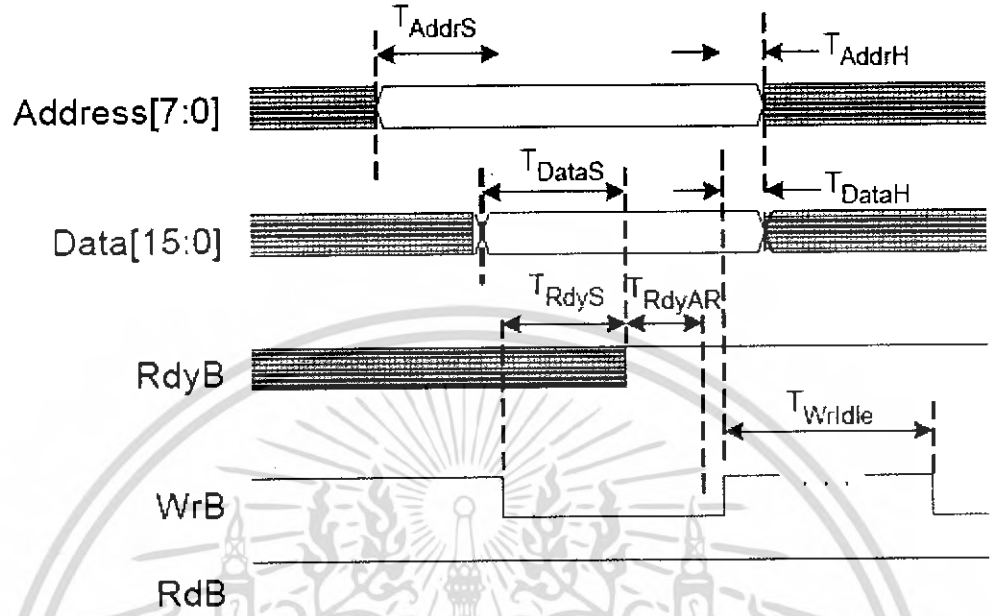
รูปที่ 3.9 ตำแหน่งของขาสัญญาณที่ connector

ตารางที่ 3.1 แรงดันสูงสุดที่แต่ละ pin connector

ชื่อสัญญาณ	คำอธิบาย	ประเภทของสัญญาณ	แรงดันสูงสุด
VCC	Device Power Supply	OUT	3.3
GND	Device Ground	OUT	0
D0 – D15	Data bus	OUT	2.0 - 0.5 / 3.3
ADR0 – ADR15	Address bus	OUT	2.0 - 0.5 / 3.3
WrB	Write signal	OUT	3.3
RdB	Read signal	OUT	3.3
RdyB	Ready signal	IN	2.0 – 0.5

3.6.4 Timing diagram ของการเขียน

1. กรณีที่สัญญาณ RdyB มีค่าเท่ากับ high ในช่วง Ready Assertion to Release Time



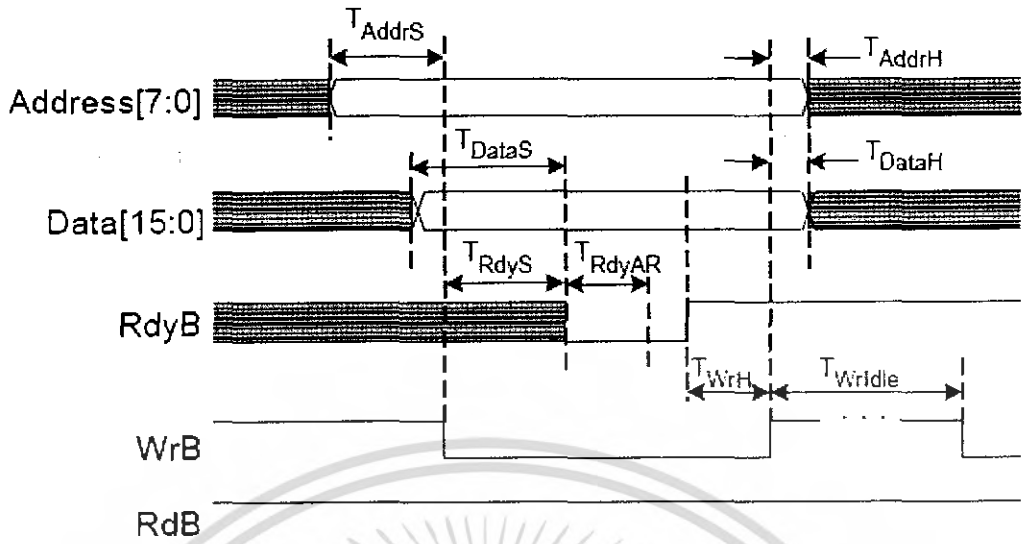
รูปที่ 3.10 รูปสัญญาณต่าง ๆ ในกรณีที่สัญญาณ RdyB มีค่าเท่ากับ high ในช่วง TRdyAR

จากรูปที่3.3 คือ สัญญาณต่าง ๆ เมื่อเขียนข้อมูล 1 word ตารางด้านล่างคือค่า setup time, hold time ของสัญญาณต่าง ๆ ที่จำเป็น

ตารางที่ 3.2 ช่วงการเขียนข้อมูล 1 word เมื่อสัญญาณ RdyB มีค่าเท่ากับ high ในช่วง TRdyAR

Symbol	Parameter	Min.	Max.	Unit
TAddrS	Address Setup Time	3.5		us
TAddrH	Address Hold Time	2.6		us
TDataS	Data Setup Time		310	ns
TDataH	Data Hold Time	25		ns
TRdyS	Ready Setup Time		240	ns
TRdyAR	Ready Assertion to Release Time	15		ns
TWrIdle	Idle Time before continue next write	6		us

2. กรณีที่สัญญาณ RdyB มีค่าเท่ากับ low ในช่วง Ready Assertion to Release Time



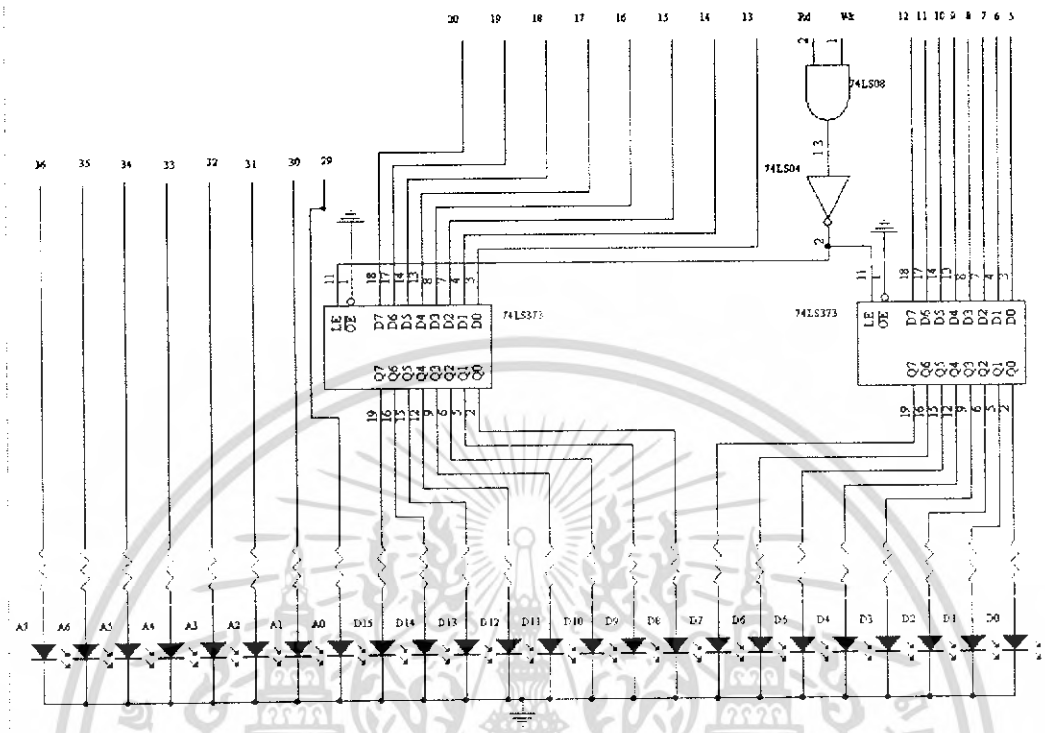
รูปที่ 3.11 รูปสัญญาณต่าง ๆ ในกรณีที่สัญญาณ RdyB มีค่าเท่ากับ low ในช่วง TRdyAR

จากรูปที่3.4 คือสัญญาณต่าง ๆ เมื่อเขียนข้อมูล 1 word ตารางด้านล่างคือค่า setup time, hold time ของสัญญาณต่าง ๆ ที่จำเป็น

ตารางที่ 3.3 ช่วงการเขียนข้อมูล 1 word เมื่อสัญญาณ RdyB มีค่าเท่ากับ low ในช่วง TRdyAR

Symbol	Parameter	Min.	Max.	Unit
TAddrS	Address Setup Time	3.5		us
TAddrH	Address Hold Time	2.6		us
TDataS	Data Setup Time		310	ns
TDataH	Data Hold Time	25		ns
TRdyS	Ready Setup Time		240	ns
TRdyAR	Ready Assertion to Release Time	15		ns
TWrH	Write Hold Time	250		ns
TWrIdle	Idle Time before continue next write	6		us

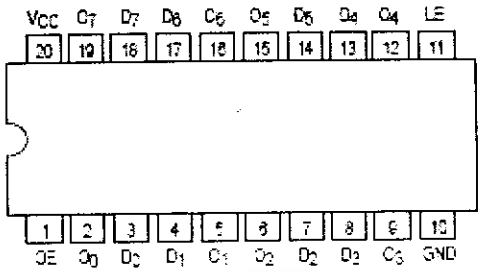
3.7 การออกแบบอุปกรณ์ที่ใช้ในการทดลอง



รูปที่3.12 แสดงวงจรบอร์ดแสดงผล

จากรูปที่ 3.12 จะแสดงวงจรของบอร์ดที่ใช้ในการแสดงผลซึ่งในวงจรนี้เราจะใช้ไอซีเบอร์ 74LS373ในการที่จะเอาข้อมูลแสดงผลเนื่องจากถ้าเราไม่นำเอาไอซีเบอร์ 74LS373มาทำการค้างสถานะไว้ข้อมูลที่ออกมาจะกระพริบแล้วก็หายไป เพราะข้อมูลที่ออกมาจากโมดูล USBนั้นไม่ได้ทำการคงค้างสถานะเอาไว้ ซึ่งทำให้เราไม่สามารถที่จะรู้ได้ว่า ข้อมูลที่ออกมานั้น ออกมาตรงกับที่เราป้อนไปหรือไม่ ซึ่งในการเอาไอซีเบอร์ 74LS373มาใช้ในการคงค้างสถานะของข้อมูลเอาไว้ แต่ในส่วนของแอดเดรสเราไม่จำเป็นต้องทำการค้างสถานะเพราะที่ตัวโมดูลจะมีการค้างสถานะไว้ให้อยู่แล้ว ซึ่งที่ขา OE เราจะป้อน 0 ให้มันเพื่อให้มันทำงานตลอด ส่วนขา enable เราจะใช้ AND GATE และ NOT GATE ในการควบคุมให้มันค้างสถานะเอาไว้

ไอซีเบอร์ 74LS373



รูปที่ 3.13 แสดงตัวถังของไอซีเบอร์อ 74LS373

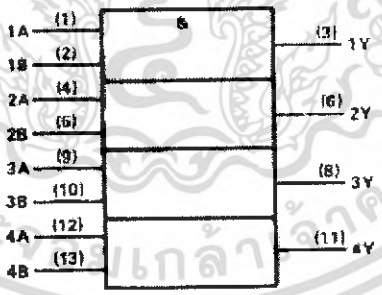
LS373

D_n	LE	OE	O_n
H	H	L	H
L	H	L	L
X	L	L	Q_0
X	X	H	Z

H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial
Z = High Impedance

รูปที่ 3.14 แสดงตารางการทำงานของ ไอซีเบอร์ 74LS373

ไอซีเบอร์ 74LS08



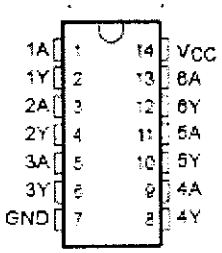
รูปที่ 3.15 แสดงโครงสร้างภายในไอซีเบอร์ 74LS08

FUNCTION TABLE (each gate)

INPUTS		OUTPUT
A	B	Y
H	H	H
L	X	L
X	L	L

รูปที่ 3.16 แสดงตารางการทำงานของไอซีเบอร์ 74LS08

ไอซีเบอร์ 74LS04



รูปที่ 3.17 แสดงตัวถังเบอร์ไอซีเบอร์ 74LS04

FUNCTION TABLE
(each inverter)

INPUT A	OUTPUT Y
H	L
L	H

รูปที่ 3.18 แสดงตารางการทำงานไอซีเบอร์ 74LS04

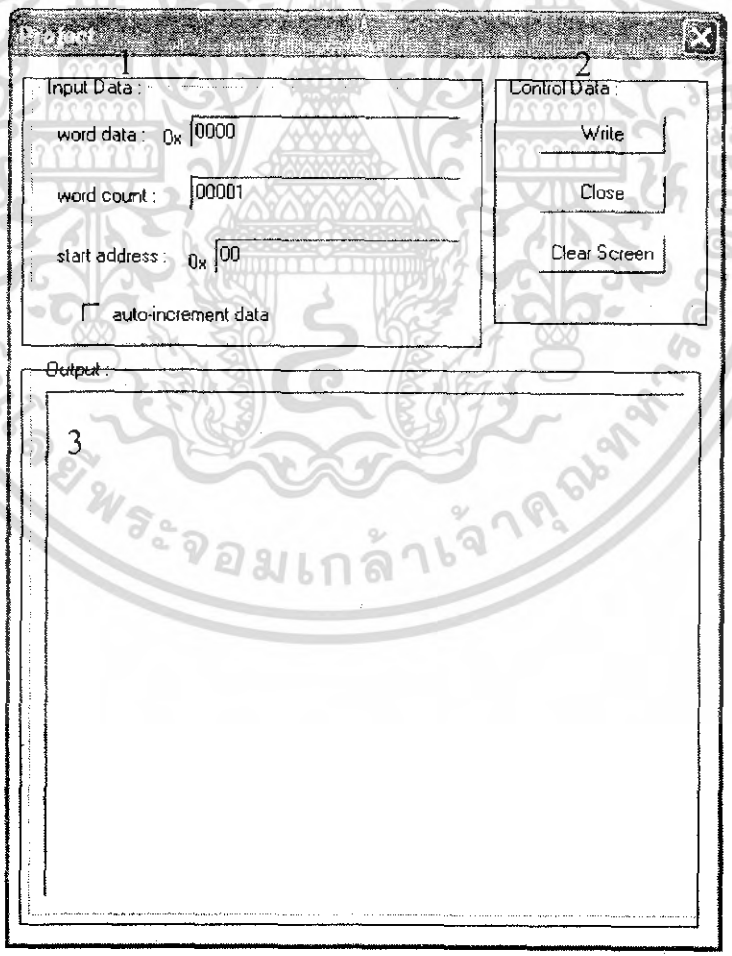
บทที่ 4

การทดลองและผลการทดลอง

ในบทนี้จะกล่าวถึงการทดลองในการเขียน driver ทั้ง 2 แบบ คือ user-mode driver และ kernel-mode driver รวมถึงการดีบักโปรแกรม kernel-mode driver ด้วยโปรแกรม debugprint

4.1 การทดลองการเขียน User-mode driver

การเขียนโปรแกรม user-mode driver จะใช้ Visual C++ 6.0 ในการเขียนและ Compile ซึ่งมีวิธีการเขียนเหมือนกับการเขียนโปรแกรมทั่วๆ ไป เพียงแต่จะต้องเรียกใช้ฟังก์ชัน Win32 API หน้าตาของโปรแกรมดังแสดงในรูปที่ 4.1



รูปที่ 4.1 User interface ของโปรแกรม

1. ส่วนที่ 1

1. Edit box "word data": ใส่ข้อมูลที่ต้องการจะเขียนไปยัง board
2. Edit box "word count": ใส่จำนวนครั้งที่ต้องการจะส่งข้อมูลออกไป
3. Edit box "start address": ใส่ address เริ่มต้นของการส่งข้อมูล
4. Radio box "auto-increment address": เลือกถ้าผู้ใช้ต้องการให้เพิ่ม address อัตโนมัติเมื่อทำการอ่านเขียน

2. ส่วนที่ 2

1. Button "Write": คลิกเพื่อเขียนข้อมูลไปยัง board
2. Button "Close": คลิกเพื่อปิดโปรแกรม
3. Button "Clear Screen": คลิกเพื่อลบข้อมูลทั้งหมดใน edit box

3. ส่วนที่ 3

1. Edit box ที่ใหญ่ที่สุด: รายงานข้อมูลที่ถูกเขียนและ address ที่ถูกเขียน

4.1.1 การทำงานของโปรแกรม

เมื่อเริ่มเปิดโปรแกรม

4.2 การทดลองและดีบัก kernel-mode driver

การเขียนโปรแกรม kernel-mode driver จะใช้โปรแกรม Editor ตัวใดก็ได้ในการเขียนแต่ในการ Compile จะใช้ Windows 2000 DDK (Device Driver Development Kit) ในการ compile และการดีบักจะใช้โปรแกรม debugprint ในการดีบัก เพราะไฟล์ที่ได้จากการ compile จะเป็นไฟล์.SYS ดังนั้นวิธีการดีบักจึงมีวิธีการเดียวคือการดู message ที่ถูกส่งออกมาจาโปรแกรม driver ของเรา

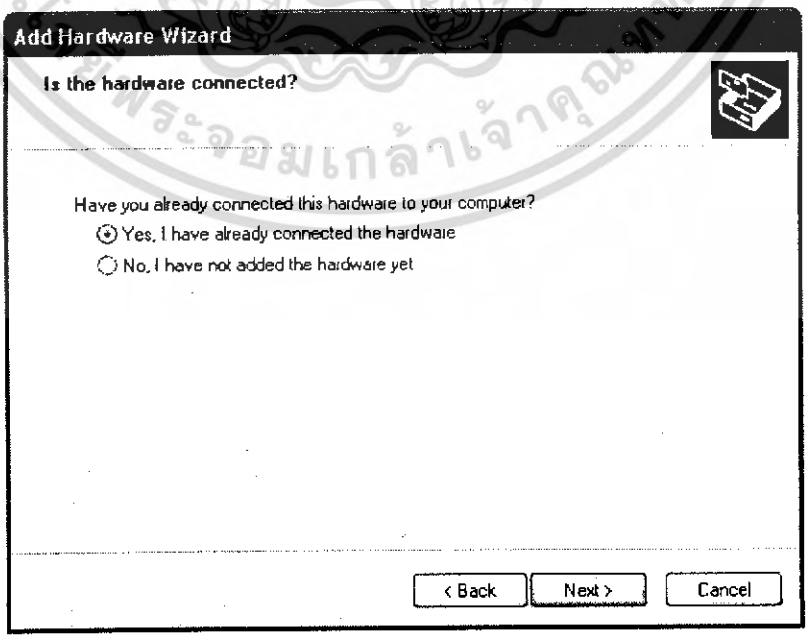
ขั้นตอนแรกให้เราทำการ ลง driver ของโปรแกรม debugprint โดยเข้าไปที่ start->Control Panel->Add Hardware จะได้หน้าต่างดังแสดงในรูป 4.2 จากนั้นให้ทำการคลิก Next จะได้หน้าต่างดังแสดงในรูป 4.3

ต่อมาให้เลือก "Yes,I have already connected the hardware" แล้วคลิก Next จะได้หน้าต่างดังแสดงในรูป 4.4

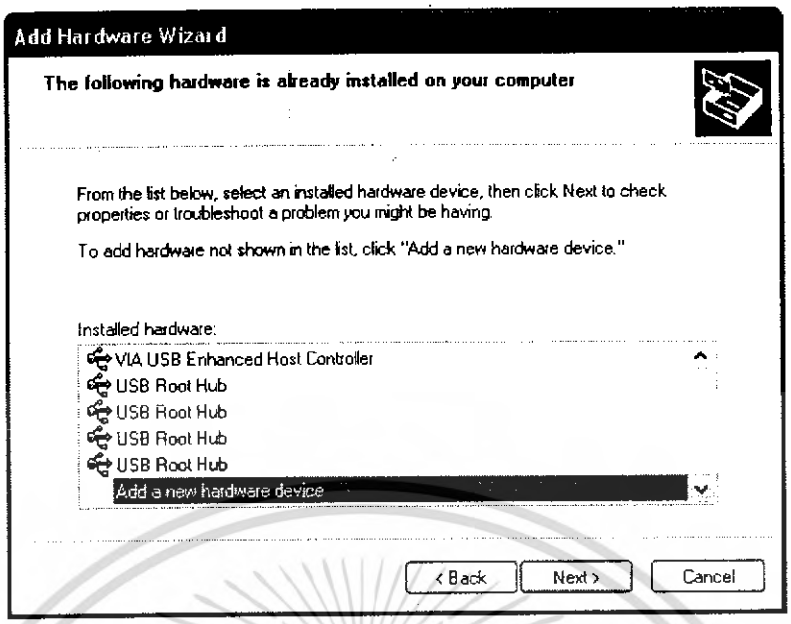
จากนั้นให้เลือก "Add a new hardware device" แล้วคลิก Next จะได้หน้าต่างดังแสดงในรูป 4.5

ให้เลือก "Install the hardware that I manually select from a list (Advanced)" แล้วคลิก Next จะได้หน้าต่างดังแสดงในรูป 4.6 เลือก "Show all device" แล้วคลิก Next จะได้หน้าต่างดังแสดงในรูป 4.7 แล้วคลิก "Have Disk" จะได้หน้าต่างดังแสดงในรูป 4.8

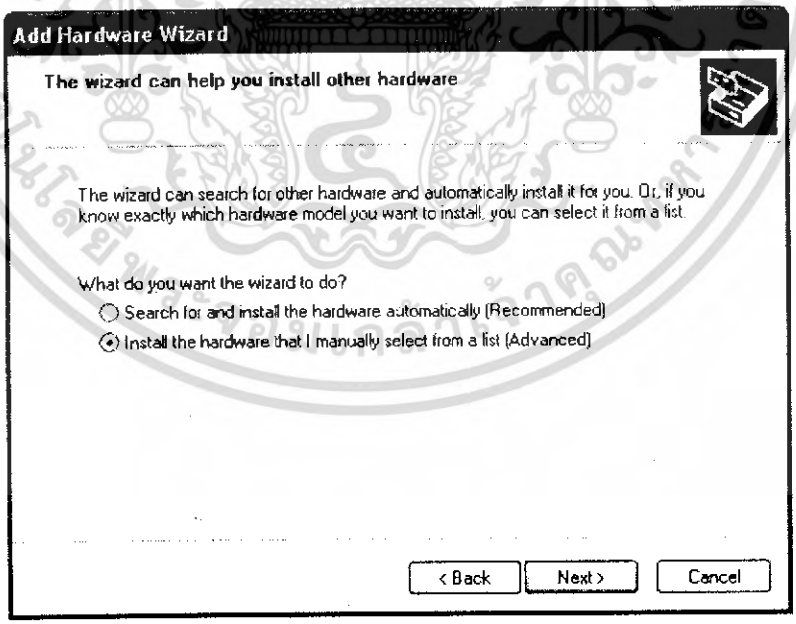
รูปที่ 4.2 การเพิ่ม driver ขั้นตอนที่1



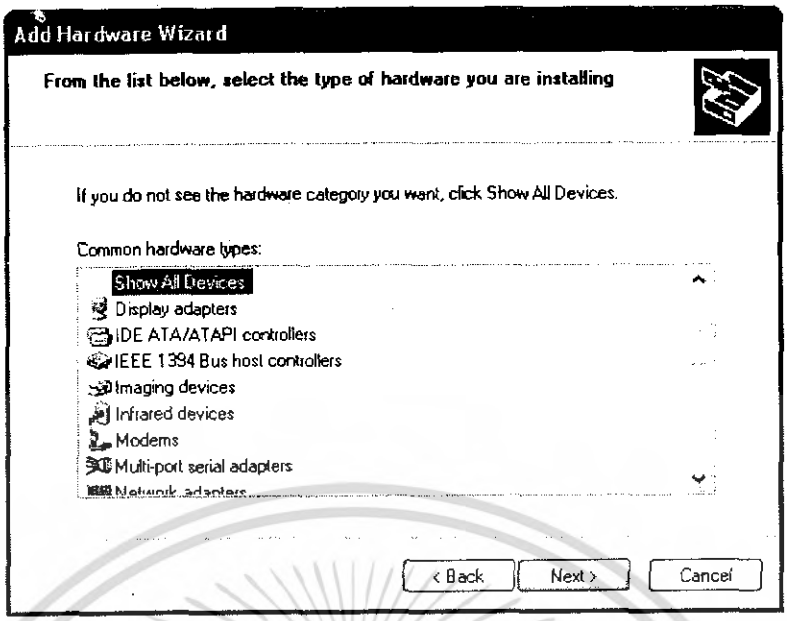
รูปที่ 4.3 การเพิ่ม driver ขั้นตอนที่2



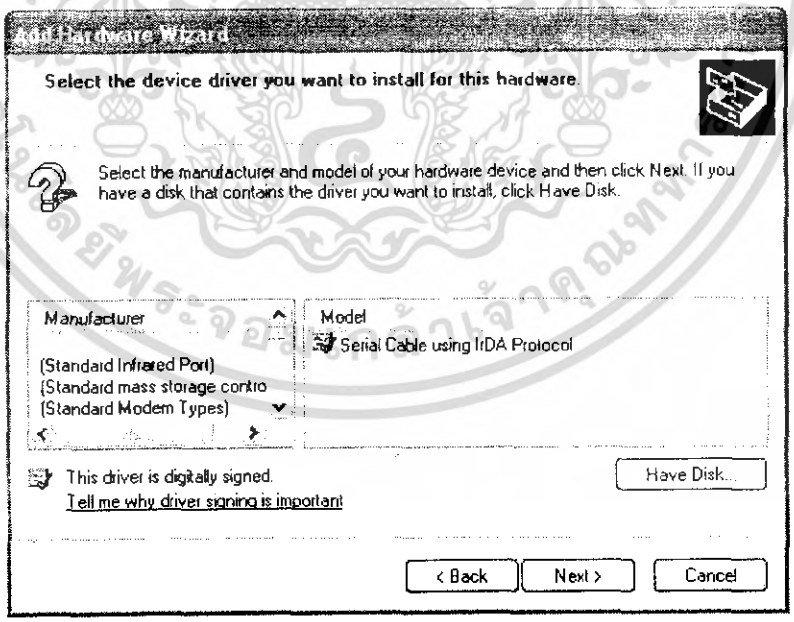
รูปที่ 4.4 การเพิ่ม driver ขั้นตอนที่3



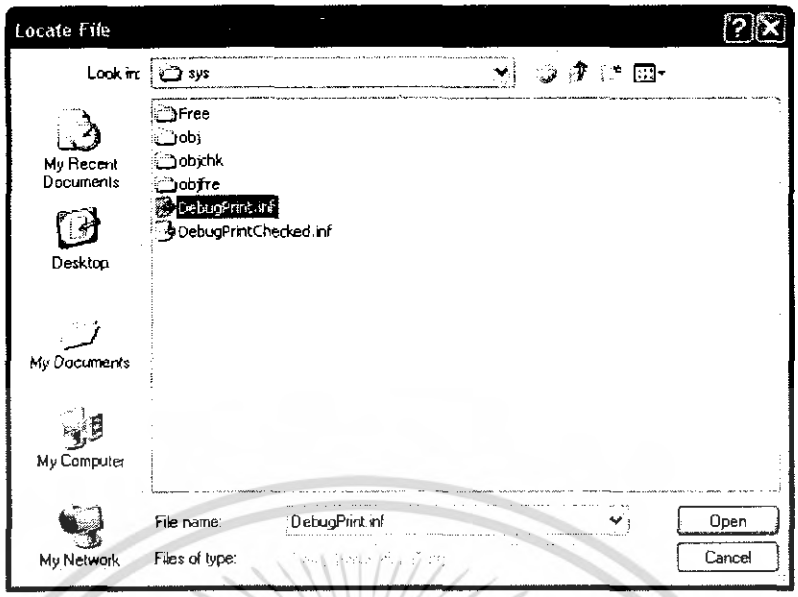
รูปที่ 4.5 การเพิ่ม driver ขั้นตอนที่4



รูปที่ 4.6 การเพิ่ม driver ขั้นตอนที่5

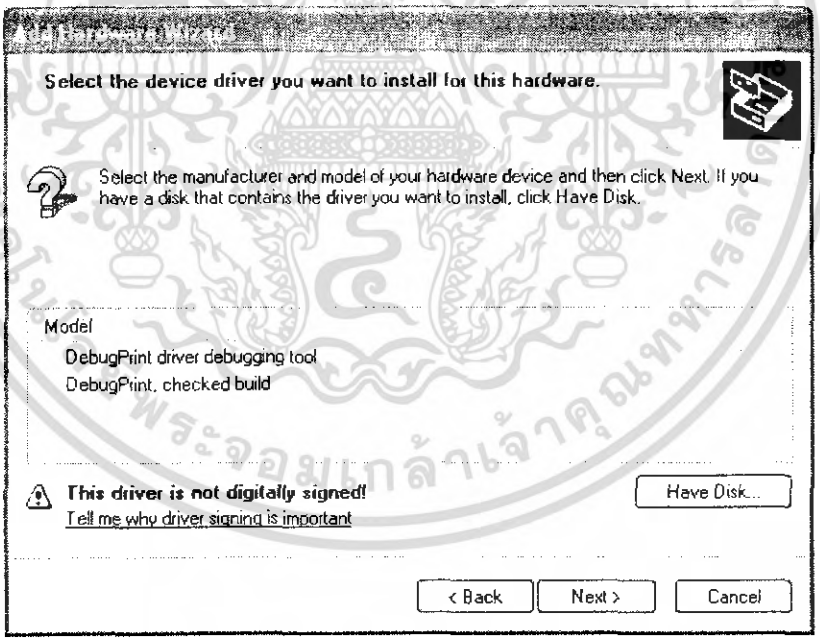


รูปที่ 4.7 การเพิ่ม driver ขั้นตอนที่6



รูปที่ 4.8 การเพิ่ม driver ขั้นตอนที่7

จากนั้นให้เลือก DebugPrint.inf คลิก Open แล้วคลิก OKจะได้หน้าต่างดังแสดงในรูป 4.8



รูปที่ 4.9 การเพิ่ม driver ขั้นตอนที่8

ให้เลือก DebugPrint, checked build คลิก Next แล้วคลิก Next จะเป็นทำสิ้นการลง driver ของโปรแกรม debugPrint

ต่อมาให้เราทำการเพิ่ม driver ของเราเข้าไปในระบบ โดยเข้าไปที่ start->Control Panel->Add Hardware

จากนั้นเมื่อเราสามารถเพิ่ม driver ของเราเข้าไปในระบบได้แล้ว เราสามารถที่จะตรวจดูได้ที่ คลิกขวาที่ My Computer->Properties ไปที่แท็ป Hardware คลิกที่ปุ่ม Device manager จะพบกับหน้าต่าง Device Manager ซึ่งจะแสดงรายละเอียดของ driver ที่ได้เพิ่มเข้าไปในระบบแล้ว

ต่อมาให้ทำการเปิดโปรแกรม debugprint ขึ้นมา การจะดู message ที่โปรแกรม debugPrint ส่งออกมานั้นเราจะต้องประกาศการใช้งานไว้ในโค้ดดังนี้

```
DebugPrintInit("String");
```

จะทำการเรียกใช้ฟังก์ชันนี้ภายในฟังก์ชัน DriverEntry เพื่อเป็นการประกาศการใช้งาน โดยค่า String ที่ใส่จะเป็นหัวเรื่องที่ใช้กำหนดว่าให้ debugPrint ส่งข้อความออกมาด้วยหัวเรื่องอะไร และเมื่อมีประกาศการใช้งานเพราะฉะนั้นเมื่อมีการ Uninstall driver ก็จะต้องทำการปิดการส่งข้อความโดยใช้คำสั่งดังนี้

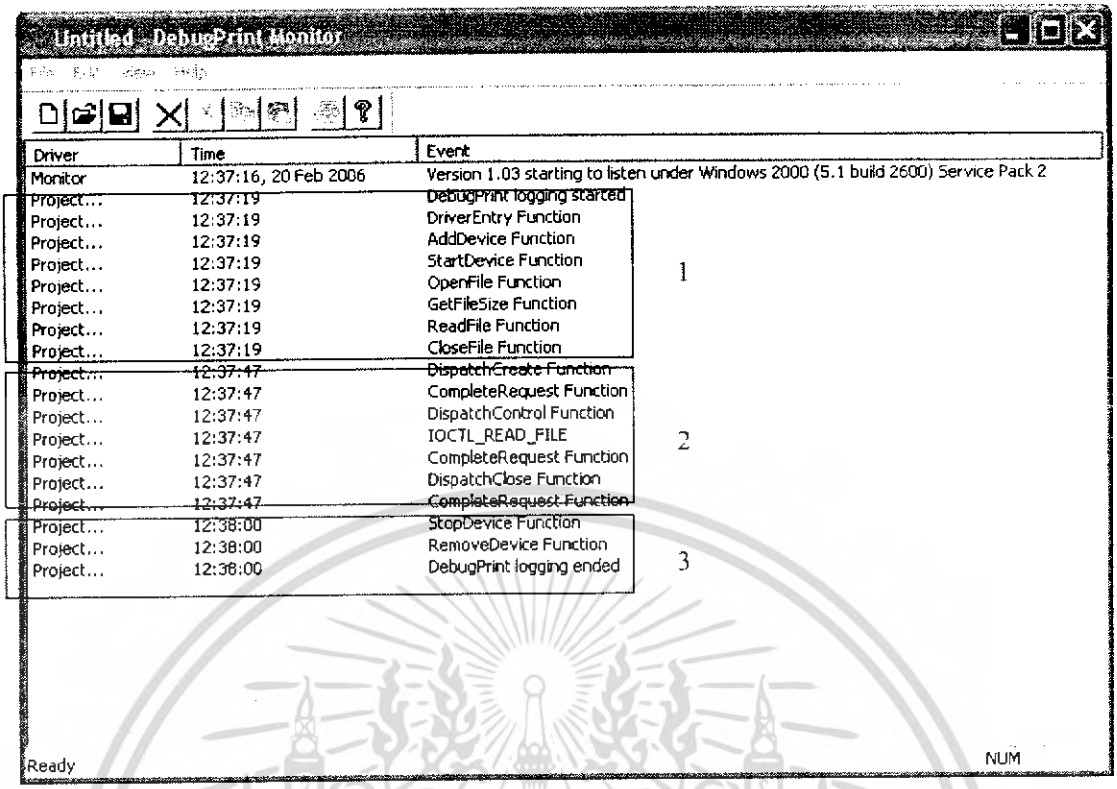
```
DebugPrintClose();
```

ซึ่งจะประกาศเรียกใช้งานที่ฟังก์ชัน DriveUnload จากนั้นถ้าภายในโปรแกรมของ kernel-mode driver ต้องการที่จะส่งข้อความออกไปให้แสดงผลที่โปรแกรม debugPrint นั้นให้ใช้คำสั่งดังนี้

```
DebugPrint("String");
```

โดยที่ string ก็คือข้อความที่เราต้องให้ส่งออกไปแสดงผลที่โปรแกรม debugPrint ประโยชน์ เพื่อดูขณะนี้ driver ของเราทำงานไปถึงที่ฟังก์ชันไหนแล้ว

จากรูปที่ 4.10 จะเป็นการดูข้อความที่ driver ของเราส่งข้อความออกมาที่โปรแกรม debugPrint โดยที่ Column ที่ชื่อ Driver จะแสดงหัวเรื่องที่เราเรียกใช้จากฟังก์ชัน DriverEntry และจะมีการแสดงเวลาที่ได้รับข้อความที่ driver ส่งเข้ามา และที่ Column Event จะแสดงข้อความที่เราได้ทำการส่งออกมาจากโปรแกรม kernel-mode driver ของเรา จากรูปที่ 4.9 จะเป็นการทดลองการอ่านข้อมูลจากไฟล์.jnk มาเก็บไว้ที่ driver โดยที่ไฟล์.jnk จะเป็นข้อความที่แสดงผลบนคอส การทดลองนี้ประโยชน์ที่ได้รับคือ รู้ว่าการทำงานของ kernel-mode driver จะมีการทำงานจากฟังก์ชันใดไปยังฟังก์ชันใดบ้าง



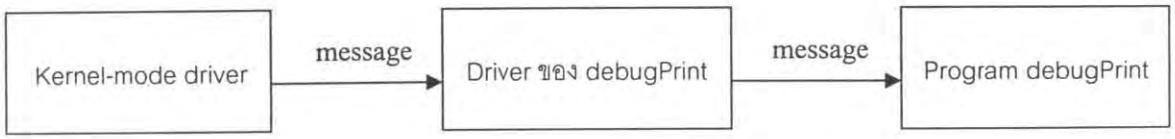
รูปที่ 4.10 การดูข้อความจากโปรแกรม debugPrint

จากรูปที่4.10 ในกรอบหมายเลข 1 เป็นขั้นตอนของการกำหนดค่าเริ่มต้นทั้งหมดของ driver โดยการทำงานจะเริ่มจาก ฟังก์ชัน DriverEntry -> AddDevice -> StartDevice -> OpenFile -> GetFileSize -> ReadFile -> CloseFile

ส่วนในกรอบหมายเลข 2 จะเป็นส่วนของการรอรับคำสั่งจาก user-mode driver เมื่อมีการใช้คำสั่ง CreateFile ที่ kernel-mode driver จะมาทำงานที่ฟังก์ชัน DispatchCreate เพื่อทำการส่ง Handle ไปให้ยัง Application จากนั้นเมื่อมีการใช้คำสั่ง DeviceIoControl ที่ kernel-mode driver จะมาทำงานที่ฟังก์ชัน DispatchControl เพื่อรับ IOCTL มาจาก user-mode driver โดยที่ IOCRL ที่รับเข้ามาคือ IOCTL_READ_FILE เพื่อทำการอ่านไฟล์ข้อความที่ driver ได้ทำการอ่านเข้ามาในขั้นตอนแรก แล้วส่งข้อความกลับไปให้ user-mode driver เพื่อส่งข้อความกลับไปให้ Application อีกทีหนึ่ง จากนั้นเมื่ออ่านไฟล์เสร็จที่ user-mode driver ก็จะทำการปิดโปรแกรมโดยใช้คำสั่ง CloseFile การทำงานที่ Kernel-mode driver ก็จะมาทำงานที่ฟังก์ชัน DispatchClose เพื่อทำการปิด Handle ของอุปกรณ์ฮาร์ดแวร์ที่ได้ทำการเปิดไว้

กรอบหมายเลข 3 เป็นการทำงานในส่วนของการ Uninstall Driver เมื่อมีการ Uninstall Driver ก็มีการทำงานที่ฟังก์ชัน Unload Driver เพื่อทำการ deallocate ตัวแปรต่างๆ ที่ได้สร้างขึ้นมาใช้ภายในโปรแกรม

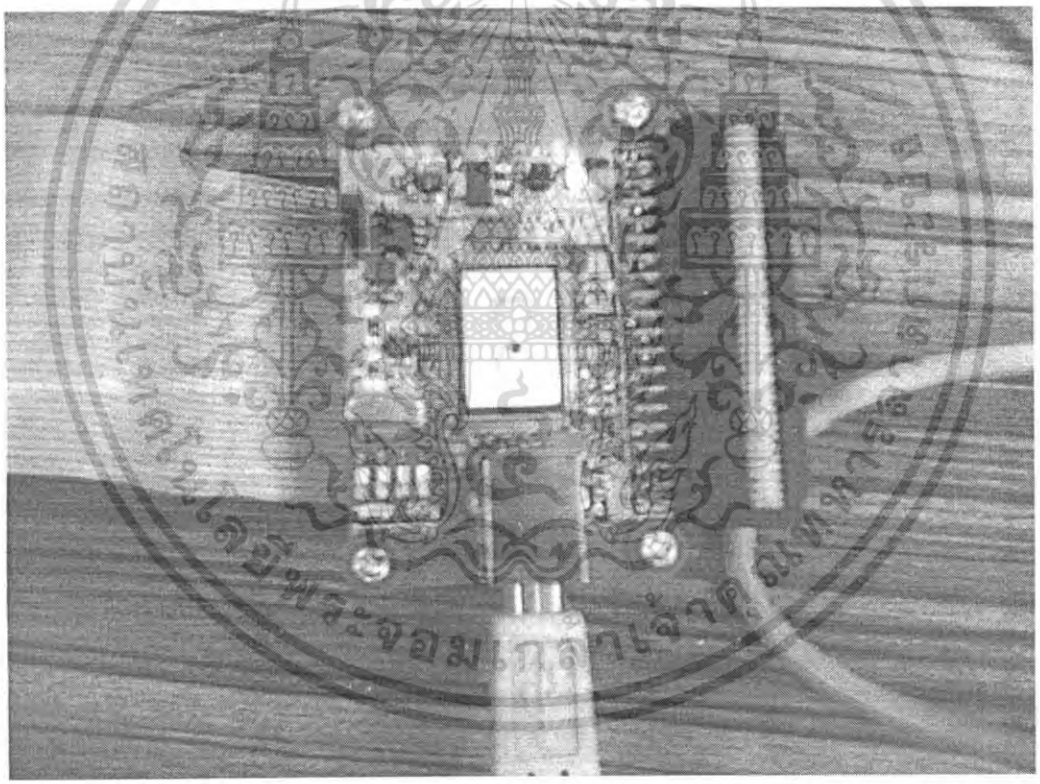
การทำงานระหว่าง kernel-mode driver กับ debugPrint ดังแสดงในรูปที่ 4.10



รูปที่ 4.11 การทำงานระหว่าง kernel-mode driver กับ debugPrint

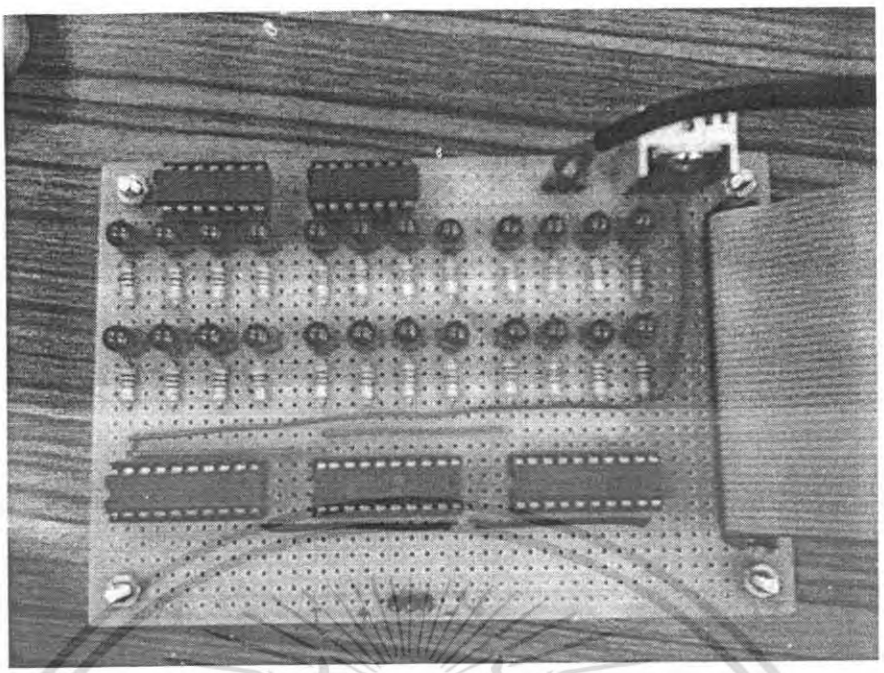
จากรูปที่ 4.11 แสดงการทำงานระหว่าง kernel-mode driver กับ debugPrint โดยที่ kernel-mode driver ต้องการส่งข้อความออกแสดงผลที่โปรแกรม debugPrint ก็จะส่งข้อความไปยัง driver ของโปรแกรม debugPrint เพื่อส่งข้อความไปแสดงผลในโปรแกรม debugPrint

4.3 การทดลองกับอุปกรณ์ฮาร์ดแวร์

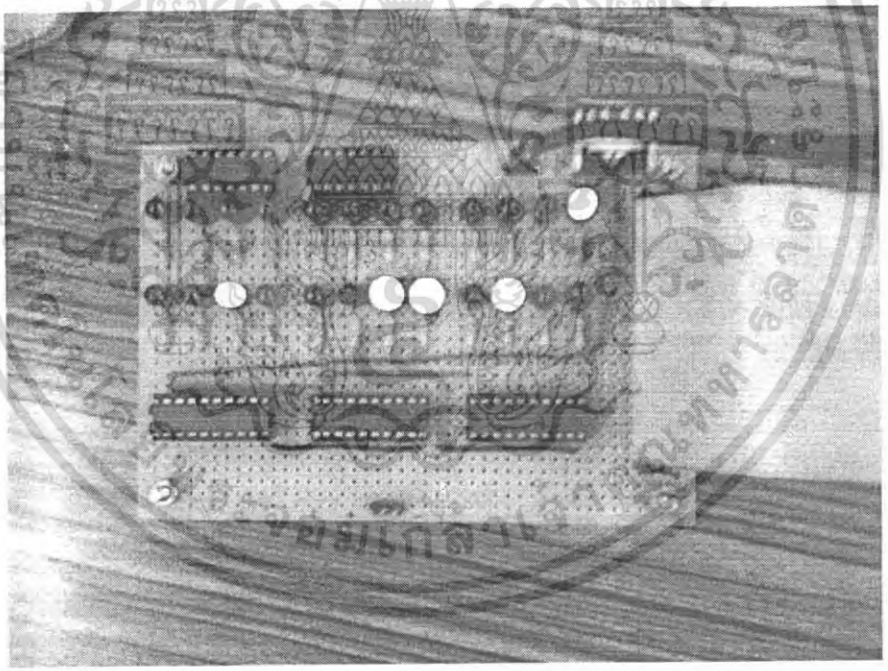


รูปที่ 4.12 แสดงโมดูล EZY I

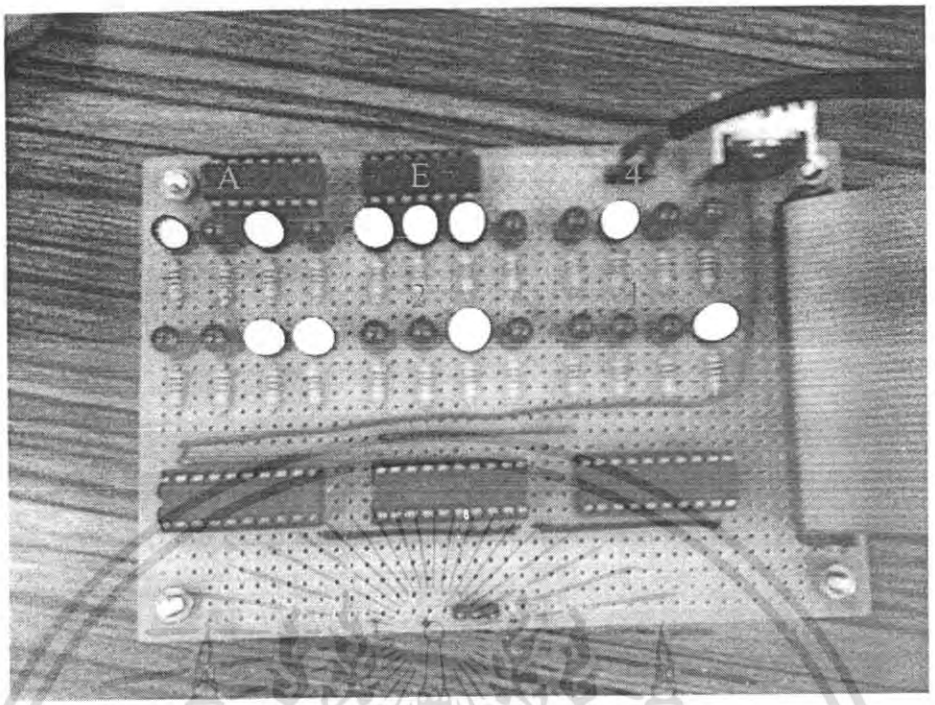
จากรูปที่ 4.12 จะแสดงตัวโมดูลที่นำมาใช้ในการรับส่งข้อมูลผ่านตัว USB ซึ่งจะมีขาสัญญาณอยู่ 40 ขา แบ่งเป็นขาแอดเดรส 8เส้น และดาต้ายึก 16 เส้น ส่วนที่เหลือก็จะเป็นส่วนของขาที่ใช้ในการควบคุมและขาจ่ายไฟเลี้ยงให้กับโมดูล



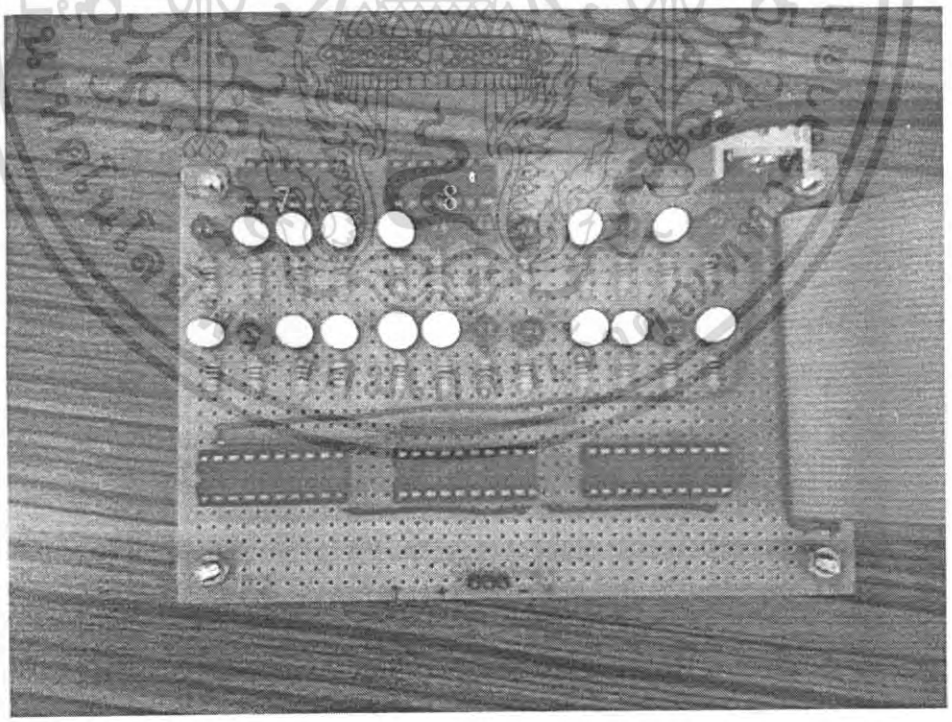
รูปที่ 4.13 แสดงโมดูลที่ใช้การทดลอง



รูปที่ 4.14 แสดงการส่งค่า 0x1234 ที่ address 0x00



รูปที่ 4.15 แสดงการส่งค่า 0x4321 ที่ address 0xAE



รูปที่ 4.16 แสดงการส่งค่า 0xABCD ที่ address 0x78

จากรูปที่ 4.13 จนถึง 4.16 จะเป็นการแสดงการทดสอบป้อนข้อมูลให้กับโมดูล USB เพื่อไปแสดงที่ตัวบอร์ดที่ใช้ในการแสดงผลออกทาง LED ในแถบซ้ายบนสองแถวแรก เราจะใช้เป็นตัวแสดงแอดเดรสที่เราป้อนเข้าไป ส่วนขวามือบนจะเป็นส่วนแสดงผลของข้อมูลซึ่งจะเป็นhigh byte และในแถวล่างก็จะเป็นส่วนของข้อมูลซึ่งเรียงจากมากไปน้อยโดยดูจากซ้ายไปขวา อย่างเช่นในตัวอย่างในรูปที่ 4.16 เราได้ทำการป้อนให้มีแอดเดรส 0x78 และดาต้าเป็น 0xABCD ซึ่งผลที่ออกมาก็จะได้ผลลัพท์อย่างในรูป

บทที่ 5
บทวิจารณ์และสรุป

5.1 บทสรุป

ในปัจจุบันระบบปฏิบัติการ Windows เป็นที่นิยมใช้กันอย่างแพร่หลาย ส่วนสาเหตุที่ทำให้ระบบปฏิบัติการ Windows เป็นที่นิยมในปัจจุบันนั้น เป็นเพราะรูปแบบการใช้งานที่ใช้งานได้ง่ายและมีหน้าตาของ User-Interface ที่สวยงามและยังมีโปรแกรมอรรถประโยชน์ต่างๆ ให้เลือกใช้งานได้อย่างมากมาย จึงทำให้ผู้ใช้งานทั่วไปจึงนิยมใช้ แต่สำหรับการที่จะทำให้ระบบปฏิบัติการ Windows ใช้งานได้ง่ายนั้น เบื้องหลังการทำงานภายใต้ User-Interface ที่สวยงาม มีการทำงานที่สลับซับซ้อนภายใต้ทำงานของตัวจัดการที่เรียกว่า "Plug and Play" จึงทำให้ระบบปฏิบัติ Windows ใช้งานได้อย่างสะดวกสบายเพียงผู้ใช้งานทั่วไปนำอุปกรณ์ฮาร์ดแวร์มาใส่แล้วติดตั้ง driver เพียงเท่านี้ผู้ใช้งานทั่วไปก็สามารถใช้งานอุปกรณ์ฮาร์ดแวร์ตัวนั้นได้แล้ว โดยที่ผู้ใช้ทั่วไปไม่ต้องมีความรู้ทางด้านโปรแกรมมิ่ง ก็สามารถควบคุมการทำงานของอุปกรณ์ฮาร์ดแวร์ตัวนั้นได้

สำหรับในโครงงานได้ทำการศึกษาและเขียน Device Driver ในส่วนของ user-mode driver ที่ใช้สำหรับการติดต่อกับโปรแกรมสำเร็จรูปที่ผู้ใช้ใช้งานโดยการทำงานของ user-mode driver ส่วนมากจะเห็นกันมากในรูปของไฟล์ .dll ที่มาพร้อมกับโปรแกรมสำเร็จรูปทั่วๆไป โดยการทำงานจะคอยรับฟังการทำงานจากโปรแกรมสำเร็จรูป แล้วตอบสนองการทำงานผ่านระบบปฏิบัติการ windows 2000 ซึ่งจะทำงานควบคู่กับ kernel-mode driver ที่เป็นส่วนที่ติดต่อกับอุปกรณ์ภายนอกอีกที การติดต่อกันระหว่าง user-mode driver กับ kernel-mode driver จะพูดคุยติดต่อกันโดย user-mode driver จะส่ง IRP ไปให้ kernel-mode driver ในการทำงานต่างๆ ก็จะมี IRP ที่เป็นเสมือน ข้อความ (messsage) ที่ user-mode driver ใช้ในการติดต่อสื่อสารกับ kernel-mode driver และการติดต่อกับอุปกรณ์ฮาร์ดแวร์ user-mode driver จะติดต่อผ่าน kernel-mode driver โดยจะส่ง IRP และส่ง IOControl Code ไปให้ kernel-mode driver เพื่อบอก kernel-mode driver ว่าต้องการที่จะติดต่อกับอุปกรณ์ฮาร์ดแวร์เพื่ออะไร จะติดต่อกับอุปกรณ์ฮาร์ดแวร์เพื่อจะอ่านค่าข้อมูลจากอุปกรณ์ฮาร์ดแวร์หรือเพื่อเขียนข้อมูลส่งไปให้อุปกรณ์ฮาร์ดแวร์แสดงผล

โครงงานนี้สามารถทำงานได้สำเร็จในส่วนของ user -mode driverโดย kernel-mode driver ยังเป็นส่วนที่มากับอุปกรณ์ฮาร์ดแวร์อยู่ และสามารถทำงานสำเร็จในส่วนของโปรแกรมที่ใช้ติดต่อกับผู้ใช้ โดยสามารถส่งข้อมูลออกไปยังอุปกรณ์ฮาร์ดแวร์ได้และสามารถกำหนดแอดเดรสที่จะส่งข้อมูลไปยังอุปกรณ์ฮาร์ดแวร์ได้

5.2 วิจารณ์สิ่งที่ได้จากโครงงาน

จากการศึกษาโครงงานนี้สิ่งที่ได้รับจากโครงงานนี้คือความรู้ความเข้าใจในระบบปฏิบัติการ Windows รวมถึงเข้าใจการทำงานภายในของระบบปฏิบัติการ Windows, เข้าใจการทำงานของ Plug and Play, เข้าใจการทำงานของระบบไฟล์และเข้าใจการเขียนโปรแกรมเพื่อติดต่อกับอุปกรณ์ฮาร์ดแวร์ที่ต้องมีการทำงานร่วมกับตัวจัดการที่เรียกว่า "Plug and Play" โดยการเขียนโปรแกรมเพื่อติดต่อกับอุปกรณ์ฮาร์ดแวร์และการทำงานกับระบบปฏิบัติการจะต้องทำอย่างระมัดระวัง เพราะหากมีการทำงานที่ไม่ดีหรือผิดพลาด อาจจะทำให้ระบบปฏิบัติการทำงานเพี้ยนได้หรือเกิดความผิดพลาดที่ระบบปฏิบัติการแสดงออกมาเช่น เกิดจอฟ้า(Blue Screen) เป็นต้น เกิดจากการไปอ้างอิงตำแหน่งใน memory ที่เป็นส่วนสงวนของระบบปฏิบัติการ Windows เช่น การอ้างอิงถึงตำแหน่งภายใน Array เกินขอบเขตที่กำหนดไว้ก็จะเกิดจอฟ้าได้ ซึ่งปัญหาการอ้างอิงถึงตำแหน่งภายใน Array เกินขอบเขตที่กำหนดไว้ ถ้าเป็นการเขียนโปรแกรมทั่วๆ ไปจะไม่การเกิดจอฟ้าเพราะระบบปฏิบัติการได้ทำการควบคุมการทำงานในส่วนนี้ไว้แล้ว แต่สำหรับการเขียน Driver นั้นเป็นการเขียนโปรแกรมที่ติดต่อกับอุปกรณ์ฮาร์ดแวร์โดยตรงซึ่งในบางส่วนอาจอยู่นอกเหนือการควบคุมของระบบปฏิบัติการได้ ดังนั้นในการเขียน driver เพื่อติดต่อกับอุปกรณ์ฮาร์ดแวร์จึงควรจะต้องใช้ความระมัดระวังเป็นพิเศษ เพราะถ้าเกิดจอฟ้าขึ้นแล้ววิธีเดียวที่จะแก้ไขได้คือ การรีสตาร์ทเครื่อง ซึ่งถ้าเกิดบ่อยๆ อาจจะทำให้เครื่องพังได้

การเขียนโปรแกรมเพื่อทำงานกับระบบหรือทำงานบน kernel space ต้องเขียนด้วยความระมัดระวังและรอบคอบ ควรจะมีการเช็ค error ทุกครั้งที่มีการใช้ฟังก์ชันเพื่อติดต่อกับระบบปฏิบัติการ windows หรือทุกครั้งที่มีการติดต่อกับอุปกรณ์ฮาร์ดแวร์ ว่าค่าที่ได้รับกลับมาเป็นที่ต้องการหรือค่าแจ้งความผิดพลาดที่ระบบส่งกลับมา ถ้าหากไม่มีการตรวจเช็คที่ดีแล้วอาจทำให้ระบบปฏิบัติการ windows ที่ใช้งานอยู่เสียหายได้ อาจถึงขั้นใช้งานไม่ได้ต้องลง windows ใหม่เพียงวิธีเดียว ในการตรวจเช็คอาจจะใช้ try-catch ในการตรวจจับก็ได้ ซึ่งเป็นฟังก์ชันที่ใช้ในการดักจับ error ที่เกิดขณะที่โปรแกรมรันอยู่

5.3 ปัญหาอุปสรรคและแนวทางในการแก้ไข

1. อุปกรณ์แต่ละตัวก็จะมี IOCTL เป็นของตัวมันเองซึ่งเป็นส่วนที่สำคัญที่ user-mode driver ใช้ในการคุยติดต่อกับ kernel-mode driver บริษัทก็จะไม่มีการเปิดเผยข้อมูลตรงส่วนนี้ให้กับลูกค้าที่ซื้อผลิตภัณฑ์ไปด้วย ดังนั้นการหาอุปกรณ์เพื่อมาใช้ทำประกอบภายในโครงงานจึงเป็นอุปสรรคเป็นอย่างมาก
2. การทำงานส่วนใหญ่ของโปรแกรมจะทำงานติดต่อกับระบบ ดังนั้นจึงต้องมีความรอบคอบและเอาใจใส่มากกว่าการเขียนโปรแกรมบน user space ปกติทั่วๆ ไป เพราะถ้าไม่มีความระมัดระวังเผลอไปใช้งานในพื้นที่ของระบบโดยไม่มีการ

ตรวจเช็คก่อน อาจจะทำให้ระบบปฏิบัติการ windows เสียหายได้ ซึ่งบ่อยครั้งจะต้องลง windows ใหม่ ทางที่ดีควรจะทำการ clone harddisk ในตอนที่ลง windows ใหม่ เพื่อเวลา windows เสียหายใช้งานไม่ได้ ก็จะได้ไม่ต้องลง windows ใหม่บ่อยๆ หรือควรจะแก้ไขวิธีการเขียนโปรแกรมให้มีความรอบคอบมากขึ้น

3. สำหรับการเขียน device driver ที่ส่วนใหญ่ทำงานบน kernel space นั้น รูปแบบการเขียนจะเป็นภาษาซีทั่วๆ ไป แต่รูปแบบฟังก์ชันจะเป็นฟังก์ชันที่ติดต่อกับระบบหรือเป็นฟังก์ชันที่ทำงานบน kernel space เสียส่วนใหญ่ ดังนั้นในช่วงแรกที่ทำการศึกษาอาจจะยากซักนิดหน่อย แต่เมื่อเริ่มเขียน เริ่มหัดบ่อยๆ ก็จะคุ้นเคย

5.4 แนวทางการพัฒนาต่อ

1. สามารถที่จะทำในส่วนของการอ่านข้อมูลเข้ามาจากอุปกรณ์ฮาร์ดแวร์โดยใช้สวิตช์เป็นอินพุทได้
2. ศึกษาการเขียนโปรแกรมบน kernel space เพื่อนำมาเขียนในส่วนของ kernel mode เพิ่มเติมจากโครงงานนี้ที่เป็นส่วนของ user mode driver
3. ศึกษาการเขียน INF file เพื่อนำมาใช้เขียน INF file เพิ่มเติมจากโครงงานนี้ให้มีความสามารถเพิ่มขึ้นมากกว่าเดิม
4. สามารถเพิ่มเติมในส่วนของฟังก์ชันการเขียนข้อมูลออกไปที่อุปกรณ์ที่ต่อพ่วงแบบหลายแอดเดรสได้
5. สามารถที่จะทำในส่วนของการอ่านข้อมูลเข้ามาจากอุปกรณ์ฮาร์ดแวร์โดยใช้สวิตช์เป็นอินพุทได้ แบบรับหลายแอดเดรสได้
6. สามารถทำการเพิ่มเติมการอ่านข้อมูลจากอุปกรณ์ฮาร์ดแวร์มาเก็บไว้ในไฟล์ได้
7. สามารถทำอุปกรณ์ฮาร์ดแวร์ขึ้นมาเพื่อใช้ในการทำโครงงานโดยสามารถใช้โครงงานนี้เป็นเอกสารอ้างอิง
8. สามารถนำไปเขียนเพิ่มเติมเพื่อใช้ติดต่อกับอุปกรณ์พอร์ตอื่นๆได้

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

บรรณานุกรม

[1] Walter Oney, 1999. **Programming the Microsoft Windows Driver Model,** Microsoft Press

[2] **Windows2000 Driver Development kit** , Microsoft Corporation , 2000

[3] Peter G. Viscarola and W. Anthony Mason, 1999. **Windows NT Device Driver Development,** Macmillan Technical Publishing

[4] Jan Axelson, 1999. **USB Complete,** Lakeview Research