โปรแกรมจัดการการใช้เครื่องพิมพ์

PRINTER USING PROGRAM FOR ADMINISTRATOR

นภาพร พันธ์โยธาชาติ
บุญทิวา บุญยกิตานนท์
อภิญญา อภิชนังกูร

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2542

PRINTER USING PROGRAM FOR ADMINISTRATOR

NAPAPORN PHANYOTHACHART
BOONTIVA BOONYAKITANON
APINYA APICHANANGKOON

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCES
FACULTY OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2000

หัวข้อปัญหาพิเศษ	โปรแกรมจัดการการใช้เครื่องพิมพ์	
	PRINTER USING PROGRAM FOR ADMINISTRATOR	
ชื่อนักศึกษา	นางสาวนภาพร พันธ์โยธาชาติ	39054627
	นางสาวบุญทิวา บุญยกิตานนท์	39054631
	นางสาวอภิญญา อภิชนังกูร	39054686
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	
สาขาวิชา	วิทยาการคอมพิวเตอร์	
อาจารย์ที่ปรึกษา	อาจารย์วีระชัย ตันยะสิทธิ์	

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร วิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์ ประจำปีการศึกษา 2542

คณะกรรมการสอบ		ลายมือชื่อ
ประธานกรรมการ	อาจารย์นันทิกา เบญจเทพานันท์	
กรรมการ	อาจารย์สิริลักษณ์ เตียพิริยะกิจ	
กรรมการและอาจารย์ที่ปรึกษา	อาจารย์วีระชัย ตันยะสิทธิ์	

(อาจารย์ไพรบูลย์ พันธรักษ์พงษ์)

หัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

หัวข้อปัญหาพิเศษ	โปรแกรมจัดการการใช้เครื่องพิมพ์	
ชื่อนักศึกษา	นางสาวนภาพร พันธ์โยธาชาติ	39054627
	นางสาวบุญทิวา บุญกิตานนท์	39054631
	นางสาวอภิญญา อภิชนังกูร	39054686
ปริญญา	วิทยาศาสตรบัณฑิต	
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์	
สาขาวิชา	วิทยาการคอมพิวเตอร์	
ปีการศึกษา	2542	
อาจารย์ที่ปรึกษา	อาจารย์วีระชัย ตันยะสิทธิ์	

บทคัดย่อ

ในปัจจุบันนี้มีการใช้ทรัพยากรในการพิมพ์เอกสารอย่างสิ้นเปลือง เนื่องจากไม่มีการจำกัดจำนวนเอกสารในการพิมพ์ ของศึกษาแต่ละคน ดังนั้นจึงมีการพัฒนาโปรแกรมควบคุมการทำงานของเครื่องพิมพ์ขึ้นมา เพื่อประโยชน์ในการตรวจสอบข้อมูลในการพิมพ์ ของนักศึกษาทุกครั้ง เพื่อทำให้นักศึกษาแต่ละคนใช้ทรัพยากรในการพิมพ์เอกสารอย่างคุ้มค่ามากขึ้น

Special Project Title	PRINTER USING PROGRAM FOR ADMINISTRATOR	
Students	Miss Napapron Phanyothachart	39054627
	Miss Boontiva Boonyakitanon	39054631
	Miss Apinya Apichanangkoon	39054686
Degree	Bachelor's Degree of Science	
Department	Mathematics and Computer Sciences, Faculty of Science	
Programme	Computer Sciences	
Academic	1999	
Special Project Advisor	Lecturer Weerachai Tanyasit	

ABSTRACT

In the present, resources for printing documents have been over-used because of the lack of limiting the number of documents that are allowed for each student to print. Therefore, we have developed a printer controller program for the benefit of checking the printing information of student each time they use the print, with the purpose for each student to use printing resources more conserving

กิตติกรรมประกาศ

ในการทำปัญหาพิเศษเรื่องโปรแกรมควบคุมการทำงานของเครื่องพิมพ์สามารถสำเร็จลุล่วงไปด้วยดี คณะผู้จัดทำต้องขอขอบคุณ อาจารย์วีระชัย ตันยะสิทธิ์ อาจารย์ที่ปรึกษาปัญหาพิเศษนี้ที่กรุณาให้คำแนะนำและเป็นที่ปรึกษาในการแก้ปัญหาต่างๆ ที่เกิดขึ้นในการดำเนินการ

นอกจากนี้คณะผู้จัดทำต้องขอขอบพระคุณ บิดา มารดา ที่ได้ให้ความสนับสนุนทางด้านกำลังและทุนทรัพย์ จนการทำปัญหาพิเศษครั้งนี้สำเร็จด้วยดี รวมทั้งเพื่อนๆ และน้องๆ ทุกคนที่ให้ความช่วยเหลือในด้านต่างๆ เกี่ยวกับปัญหาพิเศษไว้ ณ ที่นี้

คณะผู้จัดทำ

มีนาคม 2543

สารบัญ

หน้า

สารบัญ(ต่อ)

สารบัญตาราง

สารบัญรูปภาพ

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันนี้การใช้งานการพิมพ์เอกสารของนักศึกษา มักจะมีการใช้งานอย่างสิ้นเปลือง ไม่คุ้มค่า ตัวอย่างเช่น การที่นักศึกษาทำการพิมพ์เอกสารจำนวนมากโดยที่ไม่เกิดประโยชน์ทางการศึกษา แสดงให้เห็นถึงการสิ้นเปลืองทรัพยากรในการพิมพ์เอกสารเป็นอย่างมาก จึงเกิดแนวความคิดที่จะตรวจสอบจำนวนเอกสารที่นักศึกษาทำการสั่งพิมพ์ เพื่อให้นักศึกษาเกิดความระมัดระวังในการพิมพ์เอกสารมากขึ้น ซึ่งจะเป็นการทำให้นักศึกษาใช้ทรัพยากรในการพิมพ์เอกสารอย่างคุ้มค่ามากยิ่งขึ้น

1.2 ความมุ่งหมายและวัตถุประสงค์ ของการศึกษา

1.2.1 เพื่อให้การใช้ทรัพยากรในการพิมพ์เอกสารเป็นไปอย่างคุ้มค่า

1.2.2 เพื่อให้สามารถนับจำนวนแผ่นกระดาษที่มีการสั่งพิมพ์แต่ละครั้ง

1.2.3 เพื่อให้สามารถทราบถึงจำนวนแผ่นกระดาษทั้งหมดที่นักศึกษาแต่ละคนใช้

1.3 สมมติฐานของการศึกษา

สามารถทำการตรวจสอบการใช้ทรัพยากรในการพิมพ์เอกสารของนักศึกษาแต่ละคน โดยนำ Print Server เข้ามาตรวจสอบการสั่งพิมพ์เอกสารในแต่ละครั้งของนักศึกษา และทำการเก็บข้อมูลที่เกี่ยวข้องกับการพิมพ์แต่ละครั้งลงในระบบฐานข้อมูล ซึ่งจะทำให้สามารถควบคุมให้นักศึกษาใช้ทรัพยากรในการพิมพ์เอกสารให้เป็นไปอย่างคุ้มค่ามากที่สุด

1.4 ขอบเขตของการศึกษา

ในหัวข้อปัญหาพิเศษเรื่องโปรแกรมควบคุมการทำงานของเครื่องพิมพ์ มีรายละเอียดเกี่ยวกับขอบเขตของปัญหาดังนี้คือ

1.4.1 **โปรแกรม** ในส่วนนี้จะทำการนับจำนวนแผ่นกระดาษที่นักศึกษาสั่งพิมพ์แต่ละครั้ง

1.4.2 **ระบบฐานข้อมูล** ในส่วนนี้เป็นการเก็บข้อมูลและรายละเอียดต่างๆ ในการสั่งพิมพ์ แต่ละครั้ง ของนักศึกษา

1.4.3 **Network** ส่วนนี้เป็นส่วนที่ทำให้นักศึกษาสามารถสั่งพิมพ์เอกสารจาก Client เครื่องใดๆ ที่อยู่ในเครือข่ายได้

1.5 ขั้นตอนการดำเนินการวิจัย

1.5.1 ศึกษาการติดตั้ง Print Server

เป็นขั้นตอนการศึกษาการติดตั้ง Print Server และการจัดการงานพิมพ์ของ Print Server รวมทั้งรูปแบบการจัดตั้งเครื่องพิมพ์บน Print Server เพื่อให้สามารถควบคุมงานพิมพ์เอกสารได้อย่างมีประสิทธิภาพมากที่สุด

1.5.2 ศึกษาเมสเสจที่เกี่ยวข้องกับการพิมพ์เอกสาร

เป็นขั้นตอนการศึกษาและทำการวิเคราะห์เมสเสจที่เกิดของวินโดวส์ ในช่วงเวลาต่างๆ ระหว่างการพิมพ์ เพื่อให้โปรแกรมสามารถทำการประมวลผลได้ว่า เมื่อใดที่มีการสั่งพิมพ์ หรือเมื่อใดการพิมพ์ได้เสร็จสิ้นสมบูรณ์แล้ว

1.5.3 ศึกษาการนับจำนวนแผ่นกระดาษในการสั่งพิมพ์แต่ละครั้ง

เป็นขั้นตอนการศึกษาข้อมูลของเอกสารที่ผู้ใช้สั่งพิมพ์ว่ามี โครงสร้าง รายละเอียด และวิธีการที่โปรแกรมจะนำข้อมูลเหล่านี้มาใช้ เพื่อให้ได้รายละเอียดเกี่ยวกับการสั่งพิมพ์เอกสารในแต่ละครั้งของนักศึกษา

1.5.4 ศึกษาระบบฐานข้อมูล

เป็นขั้นตอนการศึกษา เกี่ยวกับระบบฐานข้อมูลซึ่งใช้ในการเก็บข้อมูลของนักศึกษา และรายละเอียดเกี่ยวกับการพิมพ์เอกสารของนักศึกษาแต่ละคน

1.5.5 ศึกษาภาษาที่ใช้ในการพัฒนาโปรแกรม

เป็นขั้นตอนการศึกษาเกี่ยวกับโครงสร้างและคำสั่งต่างๆ ของภาษาที่ใช้ในการพัฒนาโปรแกรม ทั้งในส่วนของการนับจำนวนแผ่นกระดาษที่นักศึกษาทำการสั่งพิมพ์ และการสร้างระบบฐานข้อมูล

1.5.6 ขั้นตอนการเก็บรวบรวมเอกสารและข้อมูลต่างๆ

เป็นขั้นตอนที่นำเอาเอกสารและข้อมูลที่เกี่ยวข้องมารวบรวมและใช้ประกอบการทำงาน โดยส่วนมากจะเป็นการรวบรวมจาก หนังสือ เอกสาร บทความทาง Internet และ MSDN Library รวมถึงข้อมูลที่ได้จากผู้ที่มีความรู้ในแต่ละด้าน ที่มีส่วนเกี่ยวข้องกับงานทั้งหมด

1.5.7 ขั้นตอนการวิเคราะห์และออกแบบระบบ

เป็นขั้นตอนที่ทำการนำเอาวิธีการทางคอมพิวเตอร์ ที่ได้ศึกษาจากที่เรียนมาช่วยในการวิเคราะห์และออกแบบระบบงาน โดยจะแบ่งงานออกเป็นส่วน ๆ เช่นส่วนแสดงผลลัพธ์, ส่วนการนับจำนวนแผ่นกระดาษ, ส่วนระบบฐานข้อมูล เพื่อให้ระบบทำงานได้ดีตามที่ต้องการและเป็นการกำหนดเป้าหมายในการพัฒนาโปรแกรมการทำงานด้วย

1.5.8 ขั้นตอนการพัฒนาระบบงาน

เป็นขั้นตอนการเขียนโปรแกรมตามขั้นตอนที่ได้ออกแบบไว้ในขั้นตอนการวิเคราะห์และออกแบบระบบ

1.5.9 ขั้นตอนการทดสอบโปรแกรมและปรับปรุงโปรแกรม

เป็นขั้นตอนการทดสอบโปรแกรม และบอกถึงความสามารถทั้งหมดที่เป็นไปได้ของโปรแกรมรวมถึงข้อจำกัดและขจัดปัญหาที่เกิดขึ้นกับระบบของงาน

1.5.10 ขั้นตอนการทำเอกสารประกอบ

เป็นขั้นตอนที่ทำการสร้างเอกสารประกอบโปรแกรมและเอกสารอ้างอิงในการศึกษาเพื่อทำปัญหาพิเศษ

1.6 ประโยชน์ที่คาดว่าจะได้รับ

1.6.1 สามารถนับจำนวนแผ่นกระดาษที่นักศึกษาสั่งพิมพ์ในแต่ละครั้งได้

1.6.2 สามารถตรวจสอบการใช้งานการพิมพ์เอกสารของนักศึกษาได้

1.6.3 ทำให้เกิดการใช้ทรัพยากรในการพิมพ์เอกสารอย่างคุ้มค่า

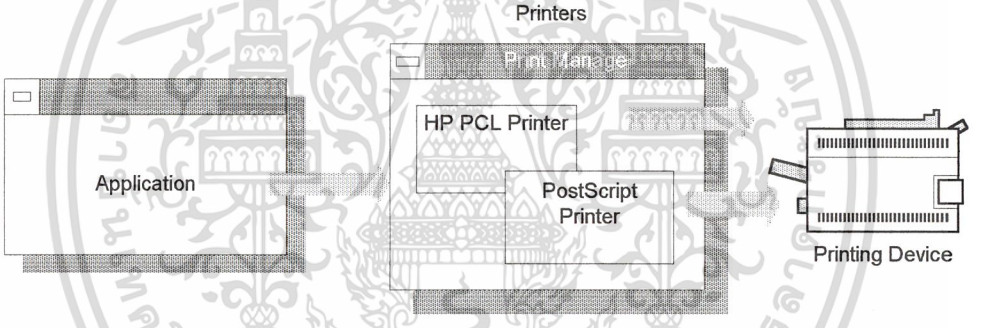
1.7 การวางแผนงาน

10 มิ.ย. – 14 มิ.ย.	ศึกษาการติดตั้ง Print Server
14 มิ.ย. – 23 ก.ค.	ศึกษาเมสเสจที่เกี่ยวข้องกับการสั่งพิมพ์
1 ก.ค. – 22 ก.ย.	ศึกษาการนับจำนวนแผ่นกระดาษในการสั่งพิมพ์
20 ก.ย. – 1 ต.ค.	ศึกษาระบบฐานข้อมูล
4 ต.ค. – 6 ต.ค	เก็บรวบรวมเอกสารและข้อมูลต่างๆ
11 ต.ค. - 5 พ.ย.	วิเคราะห์และออกแบบระบบ
1 พ.ย. - 15 ก.พ.	พัฒนาโปรแกรม
21 ก.พ. – 17 มี.ค	ทดสอบและปรับปรุงโปรแกรม
16 มี.ค. – 20 มี.ค.	ทำเอกสารประกอบ

บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 การพิมพ์งานของ Print Server

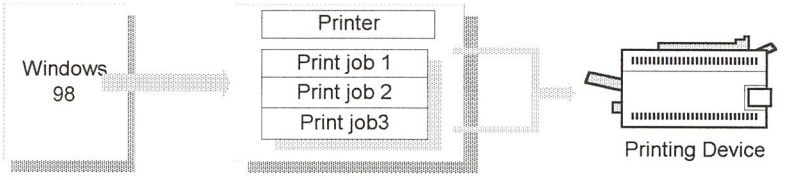
การพิมพ์งานต่างๆ บน Print Server สามารถทำได้อย่างมีประสิทธิภาพทั้งแบบ Local Printer และ Network Printer โดยจะมี 2 ส่วนในงานพิมพ์คือ Printer Device จะหมายถึงตัวเครื่องพิมพ์ที่ใช้พิมพ์งาน ส่วน Printer จะหมายถึงซอฟต์แวร์อินเทอร์เฟซ ที่ทำงานอยู่ระหว่างโปรแกรมใดๆ ที่รันบนวินโดวส์ซึ่งจะเรียกว่าแอปพลิเคชั่น กับเครื่องพิมพ์ดังรูปที่ 2.1 จะเห็นว่าเมื่อโปรแกรมแอปพลิเคชั่นต้องการจะพิมพ์งาน ก็จะต้องส่งผ่านให้ Print Manager เป็นตัวการจัดการงานพิมพ์ เพื่อส่งให้เครื่องพิมพ์
ต่อไป



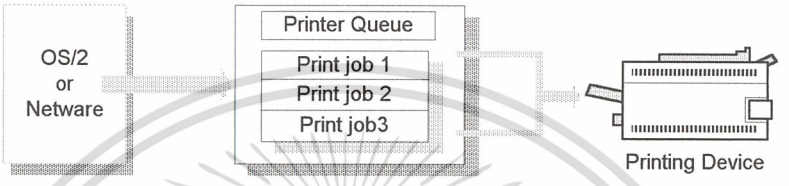
รูปที่ 2.1 แสดงการทำงานของ Printer

2.1.1 การจัดลำดับของงานพิมพ์

การพิมพ์งานบนวินโดวส์ 98 งานพิมพ์จะถูกส่งผ่านไปยัง Printer (ซอฟต์แวร์อินเทอร์เฟซ) เพื่อจัดคิวในการพิมพ์แล้วจึงส่งไปให้ Spool สำหรับส่งให้เครื่องพิมพ์ต่อไปดังรูปที่ 2.2 ก แต่สำหรับระบบปฏิบัติการ Network OS/2 และ NetWare จะส่งงานพิมพ์ไปยัง Print Queue แทนแล้วจัดการส่งงานพิมพ์ไปยังเครื่องพิมพ์ดังรูปที่ 2.2 ข



รูปที่ 2.2 ก แสดงการทำงานของ วินโดวส์ 98



รูปที่ 2.2 ข แสดงการทำงานของ OS/2 และ NetWare

2.1.2 รูปแบบการพิมพ์ของวินโดวส์ 98

การพิมพ์งานของวินโดวส์ 98 จะประกอบไปด้วย

2.1.2.1 Print Processor

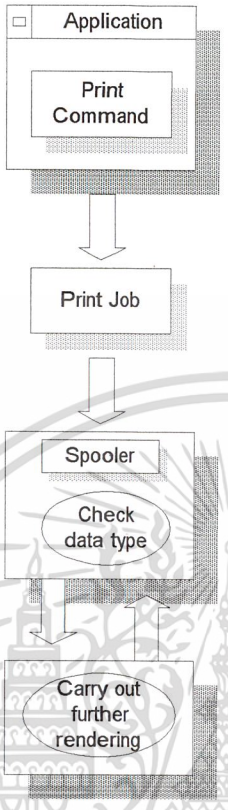
2.1.2.2 Spooler

2.1.2.3 Router

2.1.2.4 Print monitor

ซึ่งในแต่ละส่วนจะทำงานดังต่อไปนี้

2.1.2.1 Print Processor



รูปที่2.3 แสดงการทำงานของ Print Processor

เมื่อมีคำสั่งเกี่ยวกับการพิมพ์แอปพลิเคชั่นจะทำการสร้างงานพิมพ์ (Print Job) ซึ่งคือไฟล์ที่ประกอบด้วยข้อมูลที่ต้องการจะพิมพ์ และส่งให้กับ Spooler แล้ว Spooler จะตรวจสอบชนิดของข้อมูลของ งานพิมพ์ หลังจากนั้น Print Processor จะได้รับ งานพิมพ์ จาก Spooler และเมื่อ Print Processor จัดการงานพิมพ์เสร็จจะส่งงานพิมพ์กลับไปให้ Spooler อีกครั้ง

Print Processor ของ วินโดวส์ 98 (Winprnt.dll) จะจัดการกับข้อมูล 3 ชนิดด้วยกันคือ

1. Raw data เป็นข้อมูลดิบซึ่งพร้อมที่จะพิมพ์
2. Text เป็นข้อมูลธรรมดา ซึ่งจะบรรจุรหัส ASCII และรหัสควบคุมในการพิมพ์ต่างๆ
3. วินโดวส์ 98 Journal File เป็นชนิดของข้อมูลที่บรรจุ Record DDI (Device Driver Interface) ซึ่งเกี่ยวกับเอกสารจะถูกใช้โดย GDI (Graphic Device Interface) อย่างเหมาะสม Journal File จะบรรจุ Resolution (300 dpi) โดยเฉพาะและ Front Information สำหรับความละเอียดในการพิมพ์

2.1.2.2 Spooler

Spooler จะเป็นบริการสำหรับเชื่อมต่อระหว่าง แอปพลิเคชั่น และ Print Monitor โดยจะมีหน้าที่ 4 ประการคือ

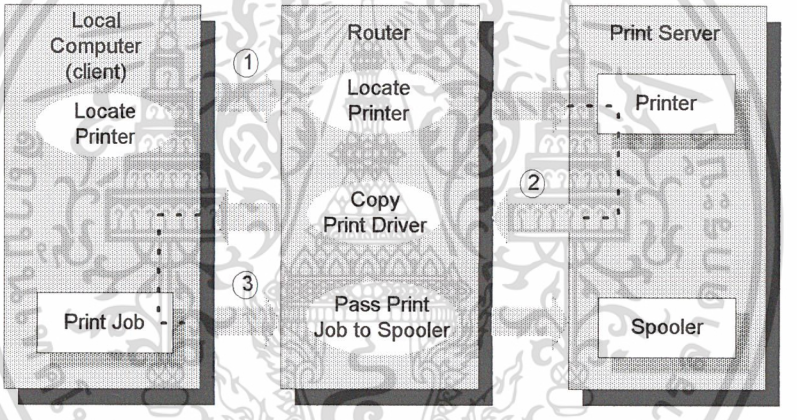
2.1.2.2.1 คอยตามในเรื่องของงานใดจะต้องพิมพ์กับเครื่องพิมพ์ใด

2.1.2.2.2 คอยตามในเรื่องของพอร์ตที่จะเชื่อมต่อกับเครื่องพิมพ์

2.1.2.2.3 วางเส้นทางของงานพิมพ์ให้ตรงกับพอร์ต รวมทั้งจัดการ Print Spool ด้วย

2.1.2.2.4 จัดลำดับก่อนหลังของงานพิมพ์

2.1.2.3 Router



รูปที่ 2.4 แสดงการทำงานของ Router

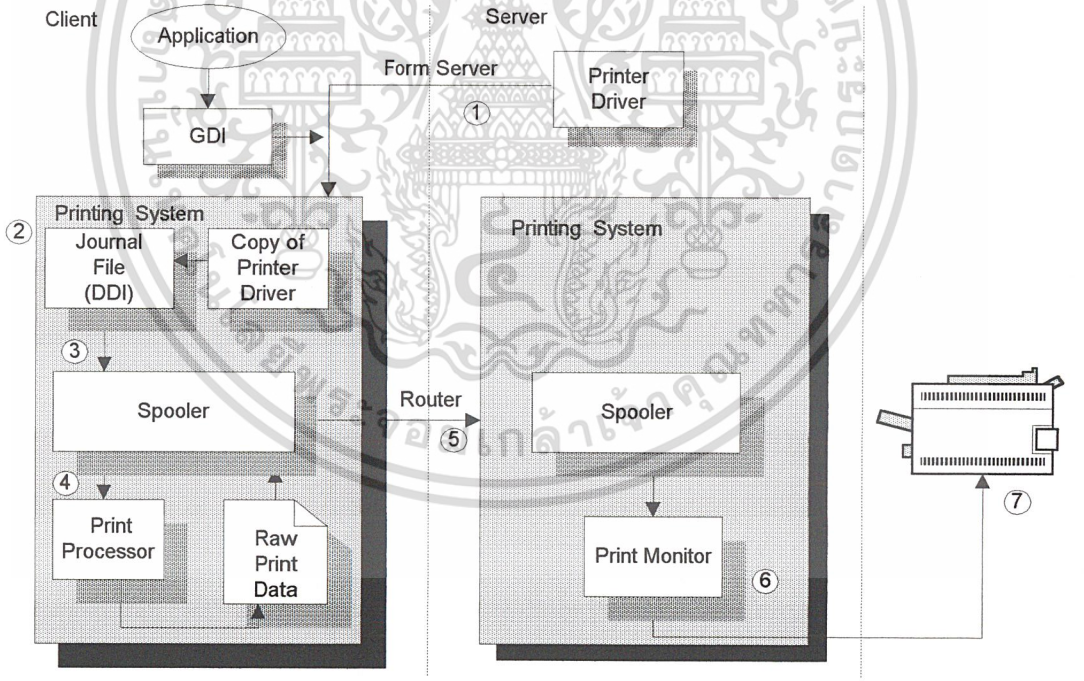
Router เป็นส่วนประกอบในการพิมพ์ จากรูป 2.4 จะเห็นว่าในขั้นที่ 1 Print Router จะทำการหาตำแหน่งที่ตั้งของเครื่องพิมพ์ที่ต้องการใช้งาน ขั้นที่ 2 จะ copy Driver ของเครื่องพิมพ์มายังคอมพิวเตอร์ท้องถิ่น และขั้นที่ 3 ก็จะส่งงานพิมพ์จากคอมพิวเตอร์ท้องถิ่นหรือ Client Spooler ไปยัง Print Server Spooler สรุปแล้ว Print Router จะมีหน้าที่ส่ง Print Information จากคอมพิวเตอร์ท้องถิ่นไปยัง Print Server

2.1.2.4 Print Monitor

ขั้นสุดท้ายก่อนที่จะพิมพ์ Print Monitor จะทำการ access หรือเข้าถึงพอร์ตเครื่องพิมพ์อย่างเหมาะสม แล้วควบคุมการไหลของข้อมูลไปยังพอร์ตเครื่องพิมพ์ ควบคุมการพิมพ์ออกไปสู่พอร์ต และยกเลิกงานพิมพ์ไปยังพอร์ตเครื่องพิมพ์นอกจากนี้ยังมีหน้าที่อื่นอีกคือ

- ตรวจดู Error message ต่างๆ เช่น " Out of Paper "
- จะแจ้งประกาศหรือส่งข่าวเกี่ยวกับการพิมพ์เสร็จ (end-of-job) Print Monitor จะรอจน กว่าเครื่องพิมพ์ได้พิมพ์งานหน้าสุดท้ายเสร็จแล้วบอก Spooler ว่างานพิมพ์เสร็จแล้ว ให้ลบงานพิมพ์นี้ออกจาก Spooler ได้
- ทำการตรวจสอบดูสถานะของเครื่องพิมพ์ เพื่อตรวจจับความผิดพลาดในงานพิมพ์ ถ้าพบความผิดพลาดหรือ Error ก็จะแจ้งให้ Spooler ทราบ

สรุปขั้นตอนการพิมพ์งานของวินโดวส์ 98



รูปที่ 2.5 แสดงขั้นตอนการพิมพ์ของ วินโดวส์ 98

จากรูป 2.5 สามารถอธิบายการทำงานเป็นขั้นตอนได้ดังนี้

1. เมื่อ Driver เครื่องพิมพ์ถูก Load ก็จะ copy Driver ของเครื่องพิมพ์มายังคอมพิวเตอร์ท้องถิ่น (ในกรณีที่ใช้งานเครื่องพิมพ์แบบรีโมต)
2. แอปพลิเคชั่น จะผลิตเอาต์พุตไฟล์ที่เป็นรูปแบบของเอกสารไฟล์นี้จะผลิตโดย GDI (Graphic Device Interface) ซึ่งบรรจุ Record DDI (Device Driver Interface) ที่เกี่ยวกับเอกสาร
3. Spoolerจะรับเอกสารมาแล้วตรวจสอบชนิดของข้อมูลก่อนส่งไปยังPrint Processor
4. Print Processor จะจัดการกับไฟล์ที่รับมาอย่างเหมาะสมขึ้นอยู่กับชนิดของข้อมูล หลังจากที่ Print Processor ได้จัดการ Render งานพิมพ์เสร็จแล้วก็จะส่งไฟล์นั้นกลับไปยัง Spooler
5. Spooler ของเครื่องคอมพิวเตอร์ท้องถิ่นจะส่งเอกสารไปยัง Spooler บน Print Server โดยผ่าน Router อีกทีหนึ่ง
6. ที่ Print Server นี้ตัว Spooler จะส่งเอกสารไปยัง Print Monitor ซึ่งควบคุมการไหลของข้อมูล - เขียนข้อมูลออกไปสู่พอร์ตปลายทางเช่น LPT1, COM, Share Printer หรือ Network Print server
7. เครื่องพิมพ์จะรับข้อมูล - เอกสารมาเพื่อทำการพิมพ์เป็นอันเรียบร้อย

2.2 การจัดการเมสเสจ กับวินโดวส์

เมสเสจเป็นอินพุตหรือข้อมูลเข้าเพียงทางเดียวของแอปพลิเคชั่น วินโดวส์เป็นตัวแทนต่อทุกๆ เหตุการณ์ทั้งหมดที่ต้องการการตอบสนอง เมสเสจเป็นตัวแปรโครงสร้างชนิดหนึ่งที่ประกอบไปด้วยส่วนค่าอ้างอิงของเมสเสจนั้นกับส่วนพารามิเตอร์ของเมสเสจ โดยที่พารามิเตอร์จะขึ้นกับเมสเสจชนิดนั้นๆ

2.2.1 การสร้างและการประมวลผลเมสเสจ

วินโดวส์จะมีการสร้างเมสเสจขึ้นทุกครั้งที่มีอินพุตเข้ามายังระบบ ไม่ว่าจะเป็นการเลื่อนเมาส์ การกดคีย์ หรืออื่นๆ เพื่อบอกแก่แอปพลิเคชั่นหรือตัววินโดวส์เองถึงเหตุการณ์ (อินพุต) ที่เกิดขึ้น โดยวินโดวส์จะนำเมสเสจที่สร้างขึ้นเหล่านี้ใส่เข้าไปในคิวของระบบ จากนั้นก็ส่งผ่านไปยังคิวของแอปพลิเคชั่นที่เหมาะสมต่อไป ลักษณะของคิวของแอปพลิเคชั่นนี้เป็นแบบเข้าก่อนออกก่อนโดยที่การเข้าก็คือ วินโดวส์ดึงเมสเสจจากคิวของระบบมาใส่ให้แก่คิวของแอปพลิเคชั่น ส่วนการออกคือแอปพลิเคชั่นเป็นตัวดึงออกไปเองโดยเรียกใช้ฟังก์ชัน Getmessage จากนั้นก็แจกแจงและส่งไปยังฟังก์ชันประจำวินโดวส์ต่างๆ ด้วยฟังก์ชัน DispatchMessage

แต่ใช่ว่ามีเพียงวินโดวส์เท่านั้นที่จะสามารถสร้างเมสเสจขึ้นมาได้ ตัวแอปพลิเคชั่นเองก็สามารถสร้างเมสเสจแล้วส่งกลับไปยังคิวของตนเองและคิวของแอปพลิเคชั่นอื่นๆ ได้เช่นกัน

แอปพลิเคชั่นจะใช้ฟังก์ชัน Getmessage ในลูปฟังก์ชัน WinMain เพื่อดึงเมสเสจออกจากคิวของตัวแอปพลิเคชั่นเอง ฟังก์ชัน GetMessage จะทำงานโดยเริ่มจากดูว่าในคิวมีเมสเสจหรือไม่หากมีก็จะดึงเมสเสจที่อยู่แรกสุดไป แต่หากไม่มีเมสเสจอยู่ในคิว ก็จะเกิดการรอเมสเสจขึ้น และปล่อยการควบคุมกลับคืนไปให้วินโดวส์ เพื่อแบ่งการควบคุมให้แอปพลิเคชั่นอื่นสามารถทำงานได้ต่อไป

เมื่อฟังก์ชัน WinMain ของแอปพลิเคชั่น ได้รับเมสเสจแล้ว ก็ต้องใช้เรียกฟังก์ชัน DispatchMessage เพื่อแจกแจงเมสเสจ และการส่งเมสเสจนั้นไปให้ฟังก์ชันประจำวินโดวส์นี้เรียบร้อย แล้วส่งการควบคุมกลับไปให้ฟังก์ชันหลักในแอปพลิเคชั่นเพื่อดึงเมสแสจในคิวมาประมวลผลต่อไป

2.2.2 การส่งและส่งผ่านเมสเสจ

ฟังก์ชัน PostMessage และ SendMessage มีหน้าที่ในการส่งเมสเสจไปยังฟังก์ชันประจำวินโดวส์ของตัวเอง หรือฟังก์ชันประจำวินโดวส์ของแอปพลิเคชั่นอื่น นอกจากนี้ยังมี

ฟังก์ชัน PostAppMessage ที่ทำงานแบบเดียวกับฟังก์ชัน PostMessage แต่เป็นการส่งผ่านโดยอาศัยแฮนเดิลโมดูลของแอปพลิเคชั่นแทน

ฟังก์ชัน PostMessage จะส่งเมสเสจผ่านไปทางคิวของแอปพลิเคชั่น (เราจะขอเรียกการส่งแบบนี้ว่า การส่งผ่าน) โดยที่ผู้ส่งจะยังไม่เสียการควบคุมและผลของเมสเสจที่ส่งไปนั้นก็จะยังไม่เกิดขึ้นจนกว่าแอปพลิเคชั่นนั้นๆ จะดึงเมสเสจที่ส่งไปนั้นขึ้นมาส่วน SendMessage จะส่งเมสเสจลัดคิวไปยังฟังก์ชันประจำวินโดวส์ที่ต้องการทันที (โดยเราขอเรียกการส่งแบบนี้ว่า การส่ง) โดยที่ไม่ต้องผ่านคิวของแอปพลิเคชั่น ซึ่งการส่งไปแบบนี้ ผู้ที่ส่งจะสูญเสียการควบคุมไปให้ฟังก์ชันประจำวินโดวส์ที่ส่งเมสเสจให้ และเมื่อฟังก์ชันทำงานเสร็จ แล้วก็จะคืนการควบคุมกลับมาแต่ถ้าเป็นค่าที่ได้กลับมาจากฟังก์ชัน PostMessage เป็นเพียงว่าสามารถส่งเมสเสจนั้นไปได้สำเร็จหรือไม่

2.2.3 ฟังก์ชันที่ใช้กับเมสเสจ

ฟังก์ชันในกลุ่มนี้มีหน้าที่ในการอ่านเมสเสจขึ้นมาจากคิว และประมวลผลเมสเสจในวินโดวส์ ฟังก์ชันที่ใช้กับเมสเสจก็จะมีตารางดังนี้

ตารางที่ 2.1 แสดงฟังก์ชันที่ใช้กับเมสเสจ

ฟังก์ชัน	หน้าที่
CallWindowProc	ส่งผ่านข้อมูลของเมสเสจไปยังฟังก์ชัน
DispatchMessage	แจกแจงเมสเสจและส่งเมสเสจไปยังฟังก์ชันประจำวินโดวส์ใดๆ
GetMessage	รับเมสเสจในช่วงของเมสเสจที่กำหนดไว้
GetMessagePost	ให้ค่าตำแหน่งของเมาส์ในขณะที่มีการรับเมสเสจล่าสุด
GetMessageTime	ให้ค่าเวลาในขณะที่มีการรับเมสเสจล่าสุด
PeekMessage	ตรวจสอบว่ามีเมสเสจในคิวหรือไม่โดยไม่ได้ดึงออกมา
PostAppMessage	ส่งเมสเสจไปยังแอปพลิเคชั่น
PostMessage	ส่งเมสเสจไปยังคิวของแอปพลิเคชั่น
PostQuitMessage	ส่ง WM_QUIT ไปยังแอปพลิเคชั่น
ReplyMessage	ตอบรับเมสเสจ
SendMessage	ส่งเมสเสจไปยังวินโดวส์ต่างๆ
TranslateAccelerator	ประมวลผลคีย์ทันใจของเมนูให้เป็นเมสเสจและส่งเมสเสจให้แก่วินโดวส์
TranslateMessage	แปลงค่ารหัสของคีย์ที่กดให้เป็นรหัสตัวอักษร
WaitMessage	ส่งการควบคุมไปให้แอปพลิเคชั่นอื่น
WinMain	เป็นจุดที่เริ่มเข้าไปทำงานของวินโดวส์แอปพลิเคชั่น

2.3 การสร้างและการจัดการวินโดวส์

ส่วนนี้จะเป็นเรื่องของการสร้าง ยกเลิก เปลี่ยนแปลง หรือดึงข้อมูลจากวินโดวส์เป็นเรื่องที่เรียกได้ว่าเป็นหัวใจของการสร้างแอปพลิเคชั่นบน วินโดวส์ เลยทีเดียว

2.3.1 คลาสของวินโดวส์

คลาสของวินโดวส์ หมายถึง ลักษณะเฉพาะที่เป็นตัวกำหนดว่าวินโดวส์นั้นจะมีรูปร่างหน้าตาเป็นอย่างไร มีความสามารถ และมีพฤติกรรมเป็นอย่างไร ก่อนที่แอปพลิเคชั่นจะสร้างวินโดวส์ขึ้นมาให้ผู้ใช้เห็นบนหน้าจอได้ต้องมีการสร้าง “คลาส” ของวินโดวส์เสียก่อน

การสร้างก็มีเพียงผ่านพารามิเตอร์ที่เหมาะสมไปให้ฟังก์ชัน Register Class โดยที่สามารถสร้างคลาสของวินโดวส์ได้มากเท่าที่ต้องการจากนั้นก็สร้างวินโดวส์ โดยการบอกว่า

วินโดวส์นั้นให้มีลักษณะเป็นไปตามคลาสใด ซึ่งคลาสหนึ่งๆ นั้น สามารถใช้โดยที่วินโดวส์ก็ได้ และคลาสของวินโดวส์ที่สร้างไว้ก็จะคงอยู่จนกว่าจะจบแอปพลิเคชั่น

แม้ว่าข้อมูลที่ต้องใส่ตอนสร้างคลาสนั้นจะมีมากมาย แต่จะมีไม่กี่ตัวที่จำเป็นจริงๆในการสร้างคลาส เช่น ชื่อคลาส ฟังก์ชันที่ใช้ในการรับเมสเสจมาประมวล และ แฮนเดิล ที่เป็นค่าคงที่ของแอปพลิเคชั่นนั้นๆ ส่วนข้อมูลตัวอื่นๆ ก็จะเป็นพวกขนาด ตำแหน่งลักษณะ cursor และอื่นๆ

คลาสของวินโดวส์สามารถแบ่งได้ตามลักษณะการสร้างและการคงอยู่ 3 ประเภทดังนี้

2.3.1.1 คลาสของระบบ

คลาสชนิดนี้ วินโดวส์จะสร้างขึ้นเมื่อเริ่มทำงาน ทุกแอปพลิเคชั่นสามารถเรียกใช้ได้ทันทีคลาสชนิดนี้ไม่สามารถถูกสร้างหรือยกเลิกโดยแอปพลิเคชั่นได้ ตัวอย่างของคลาสเหล่านี้ก็เช่น คอนโทรล edit หรือ กรอบรายชื่อ

2.3.1.2 คลาสส่วนรวมของแอปพลิเคชั่น

แอปพลิเคชั่น หรือ library สามารถสร้างคลาสที่เรียกใช้โดยแอปพลิเคชั่นอื่นๆ ได้ โดยกำหนด CS_GLOBALCLASS ในตอนสร้างคลาสโดยที่คลาสชนิดนี้จะหายไปก็ต่อเมื่อแอปพลิเคชั่น หรือ library นั้นเลิกทำงาน

2.3.1.3 คลาสของแอปพลิเคชั่น

เป็นคลาสที่พบเห็นกันมากที่สุดคือแอปพลิเคชั่นใดสร้างก็สามารถเรียกใช้งานได้ในแอปพลิเคชั่นเท่านั้น

เมื่อมีการสร้างวินโดวส์โดยระบุคลาสที่ต้องการ วินโดวส์จะไปหาในคลาสของแอปพลิเคชั่นก่อน ถ้าไม่พบก็จะไปหาในคลาสที่ใช้ได้ทุกแอปพลิเคชั่น และถ้าไม่เจออีกก็จะไปหาในคลาสของระบบ การที่วินโดวส์มีลำดับการหาคลาสอย่างนี้ ทำให้สามารถสร้างคลาสขึ้นมาทับคลาสเดิม โดยที่กระทบถึงแอปพลิเคชั่นที่ใช้คลาสเดิมนั้นอยู่

2.3.2 ส่วนประกอบของคลาสวินโดวส์

ส่วนประกอบต่างๆ ของคลาสของวินโดวส์จะเป็นตัวที่บอกลักษณะของวินโดวส์นั้น ซึ่งข้อมูลเหล่านี้จะถูกเก็บไว้ในโครงสร้างแบบ WNDCLASS จากนั้นก็จะส่งผ่านเป็นพารามิเตอร์ให้กับฟังก์ชัน Register Class ซึ่งข้อมูลเหล่านี้มีดังตารางต่อไปนี้

ตารางที่ 2.2 แสดงข้อมลในโครงสร้างแบบ WNDCLASS

ชื่อ	จุดประสงค์
ชื่อคลาส	เป็นชื่อของคลาสซึ่งแตกต่างกันไปในแต่ละคลาส
แอดเดรสฟังก์ชันประจำวินโดวส์	เป็นการกำหนดแอดเดรสของฟังก์ชันของแอปพลิเคชั่นนั้น
แฮนเดิลอินแสตนต์	เป็นตัวบอกถึงแอปพลิเคชั่นที่ขึ้นทะเบียนคลาสนั้น
เคอร์เซอร์ประจำคลาส	กำหนดรูปร่างของเคอร์เซอร์ที่ปรากฏในวินโดวส์ของคลาสนั้น
ไอคอนประจำคลาส	กำหนดไอคอน เมื่อวินโดวส์นั้นถูกลดขนาด
สีพื้นหลังของคลาส	กำหนดสีและรูปแบบของพื้นที่ใช้งาน (client area) เมื่อแสดงวินโดวส์เป็นครั้งแรก
เมนูประจำของคลาส	กำหนดเมนูของวินโดวส์ที่ไม่มีการกำหนดเมนูโดยเฉพาะ
สไตล์ของคลาส	กำหนดว่าจะทำอย่างไรเมื่อมีการย้ายวินโดวส์ เปลี่ยนขนาดวินโดวส์จะทำอย่างไรเมื่อมีการดับเบิ้ลคลิกเมาส์และอื่นๆ
พื้นที่คลาสพิเศษ	กำหนดขนาดของหน่วยความจำ (เป็นไบต์) ที่วินโดวส์ควรจองไว้ต่อจากโครงสร้างของคลาส
พื้นที่วินโดวส์พิเศษ	กำหนดขนาดของหน่วยความจำที่ วินโดวส์ควรจองไว้ต่อจากโครงสร้างวินโดวส์ที่แอปพลิเคชั่นสร้างขึ้น

2.3.3 สไตล์ของคลาส

สไตล์ของคลาสเป็นการกำหนดลักษณะของวินโดวส์เพิ่มเติม โดยสามารถกำหนดได้ครั้งละมากกว่าหนึ่งสไตล์โดยใช้โอเปอร์เรเตอร์ OR (|) สไตล์ของคลาสมีดังตารางดังต่อไปนี้

ตารางที่ 2.3 แสดงสไตล์ของคลาส

สไตล์	คุณสมบัติ
CS_BYETALIGNCLIENT	จัดวางพื้นที่ใช้งานตามไบต์ (แนวนอน)
CS_BYTEALIGNWINDOW	จัดวางวินโดวส์ตามไบต์ (แนวนอน)
CS_CLASSDC	จองคอนเท็กซ์ดิสเพลย์เพียงหนึ่งสำหรับทุกวินโดวส์ที่ใช้คลาสนี้
CS_DBCLKS	ส่งเมสเสจดับเบิ้ลคลิกไปยังฟังก์ชันประจำวินโดวส์
CS_GLOBALCLASS	กำหนดว่าคลาสนั้นสามารถถูกใช้ได้กับทุกแอปพลิเคชั่น
CS_HREDRAW	กำหนดว่าจะต้องมีการวาดวินโดวส์ใหม่ เมื่อมีการย้ายหรือเปลี่ยนแปลงขนาดที่มีผลต่อความกว้างของวินโดวส์
CS_NOCLOSE	ไม่ให้ใช้คำสั่ง Close ในเมนูของระบบ
CS_OWNDC	จองคอนเท็กซ์ดิสเพลย์สำหรับทุกวินโดวส์ที่ใช้คลาสนั้น
CS_PARENTDC	ให้คลาสของวินโดวส์นั้นใช้คอนเท็กซ์ดิสเพลย์ของวินโดวส์ parent ของวินโดวส์นั้นๆ
CS_SAVBITS	กำหนดให้มีการเก็บภาพส่วนของวินโดวส์ที่ถูกบัง เพื่อใช้ในการวาดใหม่เมื่อวินโดวส์ถูกย้าย การทำเช่นนี้จะไม่มีการส่ง WM_PAINT ไปยังวินโดวส์ (ถ้าหน่วยความจำที่เก็บรูปภาพส่วนนั้นไม่ถูกทิ้ง)
CS_VREDRAW	กำหนดว่าจะต้องมีการวาดวินโดวส์ใหม่ เมื่อมีการย้ายหรือเปลี่ยนแปลงขนาดที่มีผลต่อความสูงของวินโดวส์

2.3.4 ฟังก์ชันประจำวินโดวส์

ฟังก์ชันประจำวินโดวส์เป็นฟังก์ชันที่ทำหน้าที่ประมวลผลเมสเสจของวินโดวส์ เมสเสจจะถูกส่งมาโดยวินโดวส์ เมื่อมีอินพุต (เช่นเมาส์ คีย์บอร์ด หรือ Timer) เข้ามาหรือไม่วินโดวส์ก็ต้องการให้วินโดวส์ทำงานบางอย่าง เช่น วาดพื้นที่ใช้งานใหม่ นอกจากมีหน้าที่ในการรับอินพุตเมสเสจต่างๆแล้ว ฟังก์ชันประจำวินโดวส์ยังมีหน้าที่ในการรับข้อมูลเมื่อระบบถูกเปลี่ยนแปลง โดยแอปพลิเคชั่นอื่นๆ เช่น มีการแก้ไขไฟล์ win.ini หรือการทำตามคำร้องขอบางอย่าง เช่น การเปลี่ยนแปลงเมนูก่อนการแสดงให้ผู้ใช้เห็น หรือการรับทราบว่าขณะนี้ตัวเองได้เป็นวินโดวส์แอคทีฟแล้ว รวมทั้งข้อมูลอย่างอื่นๆ อีกมากมาย

แม้ว่าเมสเสจส่วนใหญ่จะถูกส่งมาจากวินโดวส์แต่วินโดวส์ด้วยกันเองสามารถส่งเมสเสจถึงกันได้ (หรือจากตัวเองก็ได้) ทั้งนี้ก็เพื่อจะได้ทราบความเป็นไปของวินโดวส์อื่นๆ ตามปกติ

แล้วการรับเมสเสจของฟังก์ชันประจำวินโดวส์จะเป็นไปเรื่อยๆ จนกว่าวินโดวส์นั้นจะถูกทำลาย

เนื่องจากการทำงานของแอปพลิเคชั่นนั้นขึ้นกับเมสเสจที่แต่ละวินโดวส์ได้รับ ดังนั้นตัวฟังก์ชันประจำวินโดวส์จึงเป็นส่วนที่บังคับการดำเนินไปของวินโดวส์นั้นๆ เช่น ในแอปพลิเคชั่นที่มีการเปิดไฟล์ เมื่อผู้ใช้เลือกคำสั่ง Open ฟังก์ชันประจำวินโดวส์ก็จะเป็นตัวสั่งการว่าต่อไปให้ทำการเปิดไฟล์ ตามที่ผู้ใช้ต้องการ หรือผู้ใช้มีการเลื่อนสกรอล์บาร์ฟังก์ชันประจำวินโดวส์ก็ต้องเลื่อนข้อมูลที่อยู่นอกวินโดวส์ขึ้นมาให้ผู้ใช้ดูตามต้องการ

ส่วนเมสเสจที่เข้ามายังฟังก์ชันประจำวินโดวส์แต่ไม่ได้ใช้ประโยชน์อะไรทั้งนั้น ต้องมีการส่งต่อไปให้ฟังก์ชัน DefWindowPorc เป็นตัวประมวลผลเมสเสจเหล่านั้น โดยเฉพาะอย่างยิ่งเมสเสจที่เกิดจากนอกพื้นที่ใช้งานวินโดวส์ (ซึ่งเป็นงานของวินโดวส์เช่น การคลิกที่ไตเติลบาร์หรือการลดขนาดวินโดวส์) ดังนั้นใน switch, case ของทุกฟังก์ชันประจำวินโดวส์ควรมี DefWindowProc นี้อยู่เสมอ

2.3.5 เมสเสจของวินโดวส์

เมสเสจเป็นกลุ่มของข้อมูลอย่างหนึ่งที่ถูกส่งไปยังฟังก์ชันประจำวินโดวส์ต่างๆ ที่ทำงานในวินโดวส์ขณะนั้นประกอบด้วย 4 ส่วนคือ แฮนเดิลของวินโดวส์ที่ต้องการส่งไป ตัวเมสเสจ ID พารามิเตอร์ขนด 16 บิตและสุดท้ายคือพารามิเตอร์ขนาด 32 บิต สำหรับพารามิเตอร์สองตัวหลังนั้นจะเป็นข้อมูลของอะไรก็ขึ้นกับชนิดของเมสเสจนั้น ซึ่งบางเมสเสจจะได้ใช้พารามิเตอร์ครบทั้งสองตัว

ลักษณะของฟังก์ชันประจำวินโดวส์ที่เมสเสจเป็นอาก็วเมนต์เป็นดังนี้

ตารางที่ 2.4 แสดงฟังก์ชันประจำวินโดวส์ที่เมสเสจเป็นอาก็วเมนต์

```
LONG  FAR  PASCAAL  WndProc(hWnd,wMsg,wParam,iparam )
HWND          hWnd;
WORD          wMsg;
WORD          wParam;
DWORD         lparam;
```

HWnd เป็นพารามิเตอร์ที่เก็บแฮนเดิลของวินโดวส์ที่รับเมสเสจนี้ wMsg เป็นเมสเสจที่ส่งมา wParam เป็นพารามิเตอร์เพิ่มเติมของเมสเสจที่ส่งมา โดยข้อมูล 16 บิตและ lparam เป็นพารามิเตอร์เพิ่มเติมของเมสเสจที่ส่งมา โดยเป็นข้อมูล 32 บิต ตัวอย่างของการใช้งาน

พารามิเตอร์สองตัวหลังเช่น ถ้ามีการเลื่อนเมาส์ก็จะเกิดเมสเสจ WM_MOUSEMOVE และ wParam ก็จะมีข้อมูลของปุ่มที่ถูกกด ส่วน lparam ก็จะเป็นตำแหน่งของเมาส์ที่ถูกเลื่อนมา

2.3.6 วงจรชีวิตของวินโดวส์

หน้าที่หลักของวินโดวส์ คือ การรับอินพุตและการแสดงเอาต์พุต ดังนั้นวงจรชีวิตของวินโดวส์จึงเริ่มขึ้นเมื่อแอปพลิเคชั่นต้องการรับอินพุตและเอาต์พุต และจะสิ้นสุดเมื่อไม่ต้องการหรือจบแอปพลิเคชั่น โดยปกติแล้ววินโดวส์ก็จะคงอยู่จนกว่าจะจบแอปพลิเคชั่น แต่วินโดวส์ที่ใช้งานเฉพาะบางอย่างก็จะมีชีวิตสั้นๆ เช่น วินโดวส์ของกรอบข้อความ

วินโดวส์เริ่มขึ้นจากขั้นตอนการสร้าง โดยการส่งคลาสของวินโดวส์ที่ทำการขึ้นทะเบียนไว้แล้วให้กับฟังก์ชัน Create Window จากนั้นวินโดวส์ ก็จะไปเตรียมข้อมูลเกี่ยวกับวินโดวส์ใหม่นั้นแล้วให้ค่าตัวเลขกลับมาสู่แอปพลิเคชั่นสำหรับใช้ในการอ้างอิงถึงวินโดวส์นี้ภายหลังเรียกตัวเลขนี้ว่าแฮนเดิลของวินโดวส์

เมสเสจแรกที่ถูกส่งไปยังฟังก์ชันประจำวินโดวส์ เช่น การจองหน่วยความจำ หรือเปิดไฟล์ข้อมูลเตรียมไว้ ข้อมูลที่ส่งมาด้วยก็จะอยู่ใน lparam เป็นพอยเตอร์ไปยังโครงสร้าง CREATESTRUCT ซึ่งเก็บข้อมูลเกี่ยวกับวินโดวส์นั้นอยู่ WM_CREATE และ WM_NCCREATE นี้ถูกส่งให้วินโดวส์โดยไม่ผ่านคิวของแอปพลิเคชั่น ซึ่งก็หมายความว่าจะทำงานก่อนลูปหลักของแอปพลิเคชั่น

การที่เริ่มใช้งานวินโดวส์ได้นั้นจะต้องแสดงวินโดวส์นั้นขึ้นมาเสียก่อน หากวินโดวส์ไม่ได้ถูกสร้างด้วยสไตล์ WS_VISIBLE แล้วก็ต้องใช้ฟังก์ชัน Show Window ในการแสดง แต่สำหรับวินโดวส์หลักของแอปพลิเคชั่นแล้ว ไม่ควรกำหนดให้เป็นสไตล์ WS_VISIBLE ควรใช้ Show Window แทนโดยใช้ตัวแปร nCmShow เป็นพารามิเตอร์ของฟังก์ชันเพื่อบอกว่าควรแสดงหรือไม่

เมื่อวินโดวส์สิ้นสุดการทำงานหรือจบแอปพลิเคชั่นแล้ว ก็ต้องยกเลิกหรือทำลายวินโดวส์นั้นด้วยฟังก์ชัน Destroy Window โดยฟังก์ชันจะลบวินโดวส์นั้นออกจากหน้าจอ และส่งเมสเสจ WM_DESTROY และ WM_NCDESTROY ไปยังฟังก์ชันประจำวินโดวส์ (อีกทางหนึ่งที่ฟังก์ชันของวินโดวส์ จะได้รับเมสเสจนี้คือ มีเมสเสจ WM_CLOSE ส่งเข้าไปยังฟังก์ชัน DefWindowProc) สำหรับวินโดวส์ที่เป็นวินโดวส์หลักของแอปพลิเคชั่นควรถูกทำลายเป็นวินโดวส์สุดท้าย และควรมีการบอกให้จบแอปพลิเคชั่นด้วย โดยเมื่อวินโดวส์หลักได้รับ WM_DESTROY แล้วก็ให้เรียกฟังก์ชัน PostQuitMessage เพื่อส่ง WM_QUIT ไปยังคิวของแอปพลิเคชั่นและเมื่อเมนูลูปอ่านเมสเสจนี้ขึ้นมาจะเป็นการจบแอปพลิเคชั่น

2.4 การเชื่อมต่อระบบเครือข่าย

ระบบเครือข่ายมี 2 แบบคือ

1. Peer – to - Peer
2. Client - Server

โดยแต่ละแบบมีลักษณะแตกต่างกันดังนี้

2.4.1 ระบบเครือข่ายแบบ Peer – to – Peer

เป็นระบบเครือข่ายที่คอมพิวเตอร์ทุกเครื่องมีความเสมอภาคกันหมด ไม่มีเครื่องใดที่มีหน้าที่พิเศษกว่าเครื่องอื่นๆ โดยเจ้าของเครื่องแต่ละเครื่องจะเป็นผู้ตัดสินใจเองว่า จะแบ่งปันทรัพยากรของตนเพื่อใช้งานร่วมกับผู้อื่นอย่างไร ในระบบนี้เครื่องคอมพิวเตอร์เครื่องหนึ่งๆ อาจเป็นได้ทั้ง Server โดยการอนุญาตให้ผู้อื่นเข้าใช้ File หรือเครื่องพิมพ์ของตนได้ ในขณะเดียวกันก็อาจเป็น Client ด้วยการเรียกใช้เครื่องแฟกซ์ ที่ต่ออยู่กับคอมพิวเตอร์อีกเครื่องหนึ่ง เป็นต้น เมื่อทำการเชื่อมต่อไปยังคอมพิวเตอร์เครื่องอื่นในกลุ่มงานเดียวกัน ไม่ว่ากลุ่มงานนี้จะเป็นเพียงส่วนหนึ่งในเครือข่ายขนาดใหญ่ หรือเป็นเครือข่ายเดี่ยวขนาดเล็กก็ตาม นั่นก็คือกำลังใช้งานเครือข่ายแบบ Peer – to – Peer รูปแบบการเชื่อมต่อเครือข่ายแบบ Peer – to – Peer แสดงดังรูปที่ 2.6

2.4.2 ระบบเครือข่ายแบบ Client – Server

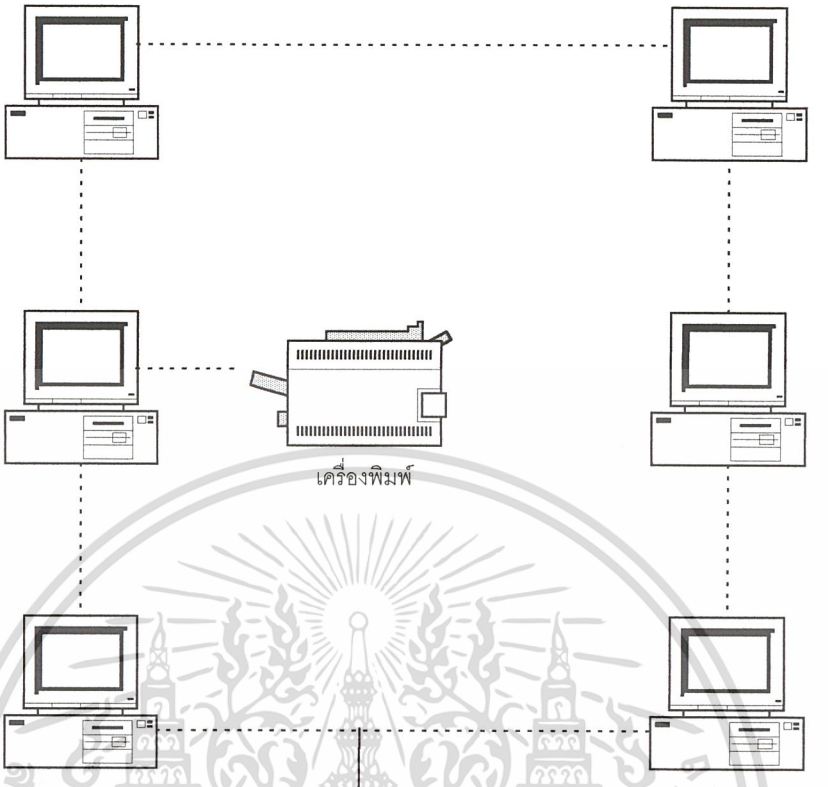
เป็นระบบเครือข่ายที่มีคอมพิวเตอร์อย่างน้อย 1 เครื่องที่ทำหน้าที่เป็น ผู้ให้บริการ ส่วนเครื่องอื่นๆ จะถือเป็น ผู้รับบริการ ในระบบนี้ Server จะทำหน้าที่เป็นศูนย์กลางการทำงานของระบบ และเป็นผู้ให้บริการงานทางด้านเครือข่ายเกือบทุกอย่าง ไม่ว่าจะเป็นการตรวจสอบชื่อผู้ใช้ และรหัสผ่าน และการควบคุมทรัพยากรของระบบ เช่น File , Disk หรือ เครื่องพิมพ์ เป็นต้น อาจกล่าวว่าเกือบทุกอย่างที่เกิดขึ้นบนเครือข่ายจะต้องผ่านไปที่ Server เสมอ ผู้ที่ทำหน้าที่ดูแล Server ก็คือ ผู้บริหารระบบเครือข่าย ซึ่งเป็นบุคคลที่กำหนดเงื่อนไขว่าจะแบ่งปันทรัพยากรของระบบอย่างไร และผู้ใช้แต่ละรายสามารถทำอะไรได้ และทำอะไรไม่ได้บ้าง (เช่น นาย ก. อาจเรียกใช้เอกสารของโครงการ A ได้ แต่ใช้เอกสารของโครงการ Bไม่ได้) นอกจากนี้บางครั้งผู้บริหารเครือข่ายก็อาจจะเป็นคนกำหนดด้วยว่า Client แต่ละเครื่องจะมีการติดตั้งอย่างไร

ในเครือข่ายแบบ Client – Server ที่มี Server หลายตัว ส่วนใหญ่แล้ว Server เหล่านี้จะถูกเชื่อมโยงถึงกัน เพื่อให้ผู้ใช้ที่ต่ออยู่กับ Server ตัวหนึ่งสามารถเข้าไปใช้บริการของ Server ตัวอื่นได้ด้วย โดย Server ทั้งหมดจะถูกจัดไว้เป็นกลุ่มๆ ที่เรียกว่า โดเมน ดังนั้นเวลาที่ log on

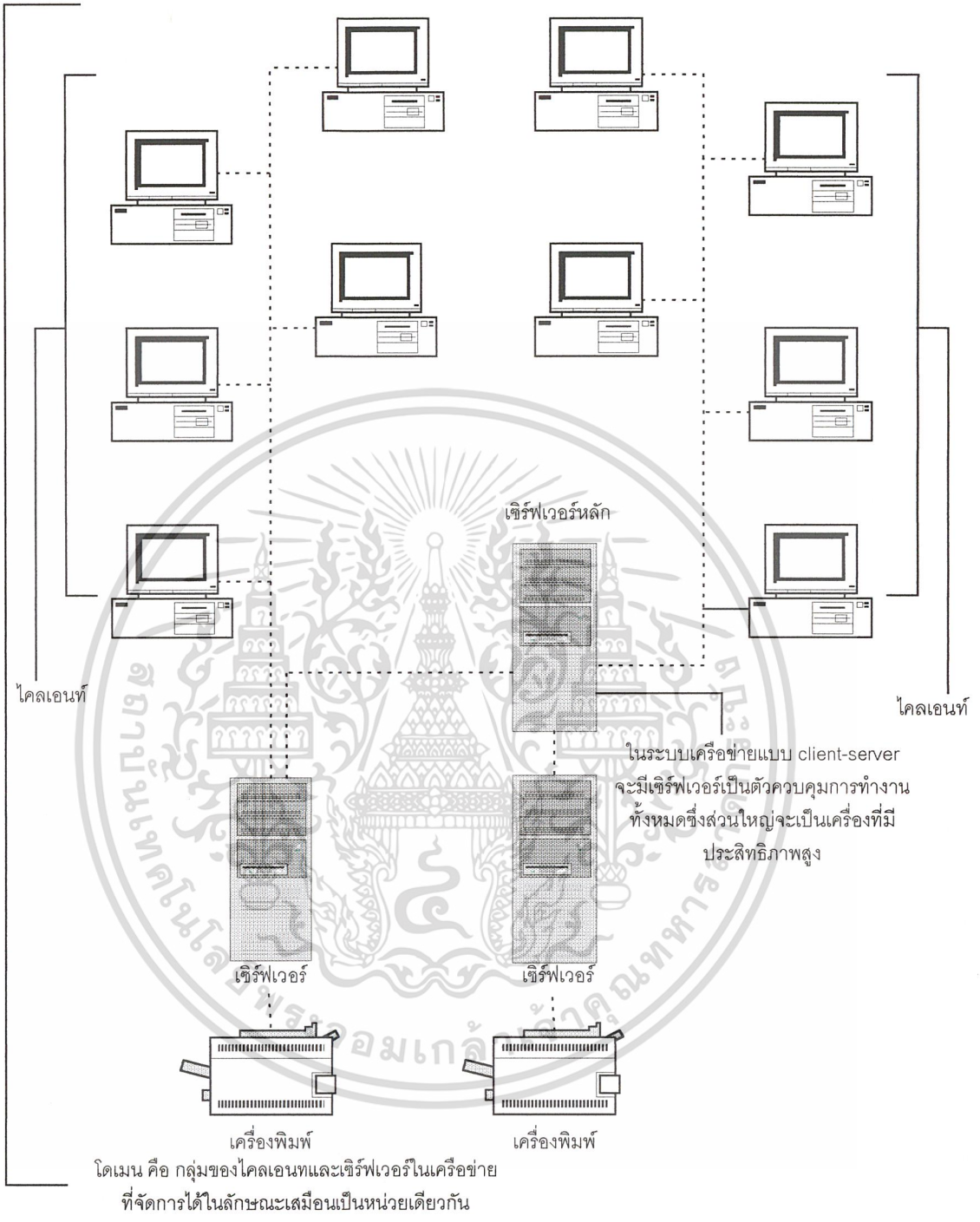
เข้าสู่ระบบนั้น (ซึ่งก็คือการ log on เข้าไปยังโดเมนนั่นเอง) จึงสามารถเข้าใช้ Server และเครื่องอื่นที่เป็นสมาชิกในโดเมนนั้น อย่างไรก็ดีเนื่องจากโปรแกรมที่ใช้ควบคุมระบบเครือข่ายมีความหลากหลายมาก และผู้บริหารเครือข่ายก็สามารถจะปรับแต่งระบบให้มีลักษณะการทำงานที่แตกต่างกันได้ ดังนั้นจึงควรศึกษาคู่มือระบบงานขององค์กร หรือขอคำปรึกษาจากผู้บริหารระบบเกี่ยวกับลักษณะ และวิธีใช้งานเครือข่ายด้วย ซึ่งรูปที่แสดงการเชื่อมต่อแบบ Client – Server จะแสดงดังรูป 2.7

ทำไมถึงต้องมีการจัดเครือข่ายเป็นแบบต่างๆกันสาเหตุสำคัญก็คือเรื่องของขนาด เพราะ Peer – to – Peer เป็นระบบที่ติดตั้งและใช้งานได้ง่ายที่สุด แต่ไม่สามารถขยายขนาดได้มาก เนื่องจากอาจก่อให้เกิดปัญหาความหนาแน่นของการรับส่งข้อมูลในเครือข่าย มีผลให้ประสิทธิภาพของเครือข่ายลดต่ำลงดังนั้นระบบนี้จึงเหมาะกับเครือข่ายขนาดเล็ก ที่มีคอมพิวเตอร์ไม่กี่เครื่องเท่านั้น ส่วน Client – Server มีโครงสร้างที่สามารถขยายตัวได้มากกว่าเพราะระบบอาจประกอบด้วย Server ที่มีประสิทธิภาพสูง มากกว่า 1 เครื่องมาเชื่อมต่อกัน ทำให้มี Client ได้เป็นจำนวนมาก อีกทั้งยังมี Software สำหรับบริหารเครือข่ายที่ถูกออกแบบเพื่อจัดการกับข้อมูลปริมาณมากๆ ที่รับและส่งในเครือข่ายโดยเฉพาะ

มาตรฐานซึ่งเป็นที่ยอมรับกันโดยทั่วไปก็คือ สำหรับสำนักงานขนาดเล็กที่มีคอมพิวเตอร์ประมาณ 10 เครื่องหรือน้อยกว่านั้น ควรใช้ระบบเครือข่ายแบบ Peer – to – Peer ส่วนองค์กรที่คอมพิวเตอร์มากกว่า 10 เครื่องขึ้นไป ควรใช้เครือข่ายแบบ Client – Server โดยแบ่งกลุ่มผู้ใช้ออกเป็นหลายๆ กลุ่มงานตามลักษณะงาน



รูปที่ 2.6 แสดงการเชื่อมต่อเครือข่ายแบบ Peer – to - Peer



รูปที่ 2.7 แสดงการเชื่อมต่อเครือข่ายแบบ Client - Server

บทที่ 3
วิธีดำเนินการวิจัย

3.1 วิธีที่ใช้ศึกษาค้นคว้า

ทำการวิจัยแบบทดลองโดยจะนำข้อมูลที่ได้จากการค้นคว้ามาทำการโปรแกรมเพื่อทดสอบผลลัพธ์ ให้เป็นไปตามแนวทางที่วางไว้

3.2 เครื่องมือและวิธีการ

3.2.1 เครื่องมือและอุปกรณ์

3.2.1.1 เครื่องคอมพิวเตอร์สำหรับเป็น Client อย่างต่ำ 1 เครื่องขึ้นไป

3.2.1.2 เครื่องคอมพิวเตอร์สำหรับเป็น Server 1 เครื่อง

3.2.1.3 เครื่องพิมพ์จำนวน 1 เครื่อง

3.2.1.4 โปรแกรม Visual C++

3.2.1.5 โปรแกรม Visual Basic

3.2.1.6 โปรแกรม Microsoft Access

3.2.2 วิธีการ

3.2.2.1 ติดตั้งโปรแกรม วินโดวส์ 98 สำหรับเครื่องที่เป็น Server และ Client

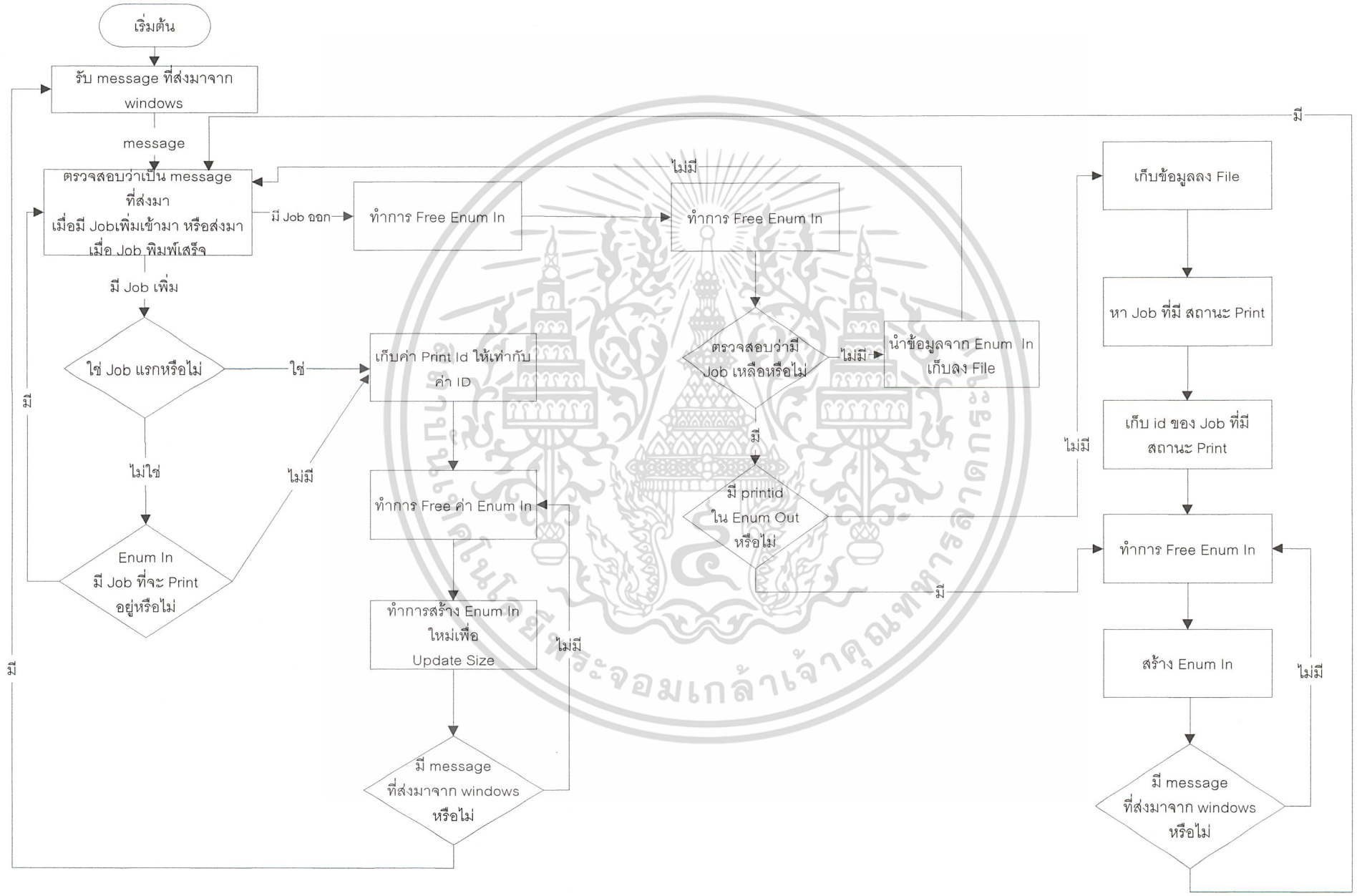
3.2.2.2 ทำการโปรแกรมในส่วนการนับแผ่นกระดาษและ User Interface ด้วย Visual C++

3.2.2.3 ทำการโปรแกรมในส่วนควบคุมฐานข้อมูลด้วย Visual Basic

3.2.2.4 ทำการสร้างฐานข้อมูลด้วย Microsoft Access

3.2.3 การวิเคราะห์การขั้นตอนการทำงานของโปรแกรม

ขั้นตอนการทำงานของโปรแกรมจะเป็นไปตาม Flow Chart ดังรูปที่ 3.1



รูปที่ 3.1 แสดงขั้นตอนการทำงานของโปรแกรม

บทที่ 4
ผลการทดลองหรือการวิเคราะห์ข้อมูล

4.1 เมสเสจ WM_SPOOLERSTATUS

จากการศึกษาข้อมูลเกี่ยวกับเมสเสจของวินโดวส์ ในส่วนที่เกี่ยวข้องกับการพิมพ์ จะพบว่าระหว่างการพิมพ์เอกสารจะมีเมสเสจที่เกิดขึ้นคือ เมสเสจ WM_SPOOLERSTATUS ซึ่งเป็นเมสเสจที่ส่งมาจาก Print Manager เมื่อมีงานถูกส่งเข้ามา หรือถูกย้ายออกไปจาก Print Manager queue โดยจะมีค่าพารามิเตอร์ 2 ค่าดังนี้

fwJobStatus

เป็นค่าของ wParam ใช้ระบุ PR_JOBSTATUS flag

cJobsLeft

เป็นค่า word ต่ำของ lparam ใช้ระบุถึงจำนวนงานที่ยังเหลืออยู่ใน Print Manager queue

จากการวิเคราะห์เมสเสจ WM_SPOOLERSTATUS พบว่าโปรแกรมจะสามารถแยกแยะได้ว่าเมสเสจใด เป็นเมสเสจที่เกิดจากการสั่งพิมพ์และเมสเสจใดเป็นเมสเสจที่เกิดจากการพิมพ์เสร็จ โดยดูจากพารามิเตอร์ lparam ดังนี้

1. เมื่อมีเมสเสจ WM_SPOOLERSTATUS แล้วพารามิเตอร์ lparam มีค่าเพิ่มขึ้นแสดงว่าเมสเสจนี้เป็นเมสเสจที่เกิดจากการสั่งพิมพ์
2. เมื่อมีเมสเสจ WM_SPOOLERSTATUS แล้วพารามิเตอร์ lparam มีค่าลดลงแสดงว่าเมสเสจนี้เป็นเมสเสจที่เกิดจากการพิมพ์เสร็จ

4.2 โครงสร้างแบบ J0B_INFO_2

จากการศึกษาเกี่ยวกับรายละเอียดของเอกสารที่ถูกสั่งพิมพ์พบว่า ข้อมูลทั้งหมดจะถูกเก็บอยู่ในรูปแบบโครงสร้างแบบ JOB_INFO_2 ซึ่งมีรายละเอียดดังนี้คือ

ตารางที่ 4.1 แสดงรปแบบโครงสร้างของ JOB INFO 2

ชื่อ	หน้าที่
JobId	ระบุถึงหมายเลขของ Job
PPrinterName	เป็น pointer ชี้ไปยัง null-terminated string ซึ่งระบุชื่อของ printer สำหรับแต่ละ Job ที่อยู่ในคิว
PMachineName	เป็น pointer ชี้ไปยัง null-terminated string ซึ่งจะระบุชื่อของเครื่องที่สั่งพิมพ์งานนั้นๆ
PUserName	เป็น pointer ชี้ไปยัง null-terminated string ซึ่งจะระบุชื่อของผู้ใช้ที่สั่งพิมพ์งานนั้นๆ
PDocument	เป็น pointer ชี้ไปยัง null-terminated string ซึ่งจะระบุชื่อของงานที่สั่งพิมพ์ (MS-WORD : " Review.doc ")
PNotifyName	เป็น pointer ชี้ไปยัง null-terminated string ซึ่งจะระบุชื่อของผู้ใช้ที่ถูกแจ้งเมื่อ Job ถูกพิมพ์หรือเมื่อ error เกิดขึ้นขณะกำลังสั่งพิมพ์งาน
PDatatype	เป็น pointer ชี้ไปยัง null-terminated string ซึ่งจะระบุชนิดของข้อมูลที่ใช้ใน record ของงานพิมพ์
PPrintProcessor	เป็น pointer ชี้ไปยัง null-terminated string ซึ่งจะระบุชื่อของ print processor ซึ่งถูกใช้พิมพ์งาน
PParameters	เป็น pointer ชี้ไปยัง null-terminated string ซึ่งจะระบุ print processor parameter
PDriverName	เป็น pointer ชี้ไปยัง null-terminated string ซึ่งจะระบุชื่อของ printer driver ซึ่งถูกใช้ process งานพิมพ์
PDevMode	เป็น pointer ชี้ไปยังโครงสร้างของ DEVMODE ซึ่งใส่ device-initialization และ environment data สำหรับ printer driver
PStatus	เป็น pointer ชี้ไปยัง null-terminated string ซึ่งระบุสถานะของ งานพิมพ์ โดยสมาชิกนี้ควรจะถูกเช็คระหว่าง Status และ ถ้า pStatus เป็น null สถานะที่ถูกกำหนดโดย content ของ สมาชิกของสถานะ
PSecurityDescriptor	ค่าของสมาชิกเป็น null การตรวจสอบและติดตั้งของความปลอดภัยของ documentไม่สนับสนุน release นี้
Status	ระบุสถานะของ job สมาชิกเหล่านี้สามารถเป็นหนึ่งหรือมากกว่าของค่าดังนี้

ตารางที่ 4.1(ต่อ) แสดงรปแบบโครงสร้างของ JOB INFO 2

	JOB_STATUS_PAUSED JOB_STATUS_ERROR JOB_STATUS_DELETING JOB_STATUS_SPOOLING JOB_STATUS_PRINTING JOB_STATUS_OFFLINE JOB_STATUS_PAPEROUT JOB_STATUS_PRINTED
Priority	ระบุถึงลำดับความสำคัญของงาน ซึ่งสมาชิกตัวนี้ สามารถมีค่าได้ 1 ค่า ตามข้างล่างนี้ หรืออยู่ในช่วง 1 ถึง 99 (MIN_PRIORITY ถึง MAX_PRIORITY) Value — Meaning MIN_PRIORITY — มีลำดับความสำคัญต่ำสุด MAX_PRIORITY — มีลำดับความสำคัญสูงสุด DEF_PRIORITY — เป็นค่า default
Position	ระบุตำแหน่งของ job ใน print queue
StartTime	ระบุเวลาเริ่มแรกสุดของ job ถูกพิมพ์
UntilTime	ระบุเวลาเริ่มท้ายสุดของ job ถูกพิมพ์
TotalPages	ระบุจำนวนหน้าที่ต้องการสำหรับ job
Size	ระบุ size (bytes) ของ job
Submitted	ระบุเวลาเมื่อ job ถูก submit
Time	ระบุเวลาทั้งหมด (วินาที) ที่ job พิมพ์ไปทั้งหมด
PagesPrinted	ระบุจำนวนหน้าซึ่งถูกพิมพ์

จากการวิเคราะห์สมาชิกของโครงสร้างแบบ JOB_INFO_2 จะพบว่าข้อมูลที่โปรแกรมต้องทำการเก็บจะมีเพียง pDocument, pUsername, Size

4.3 API Function EnumJob()

จากการศึกษาเกี่ยวกับนำข้อมูลของโครงสร้างแบบ JOB_INFO_2 มาใช้ในโปรแกรมพบว่า จะต้องทำการเรียก API Function ชื่อ EnumJob() เพื่อใช้ในการจองเนื้อที่สำหรับเก็บโครงสร้างแบบ JOB_INFO_2

4.4 Date & Time

ข้อมูลเกี่ยวกับวันที่ และเวลาที่เอกสารพิมพ์เสร็จเป็นข้อมูลที่นำมาจาก วันที่ และเวลาปัจจุบันของเครื่องที่โปรแกรมทำงานอยู่

4.5 Database

เมื่อสามารถเก็บข้อมูลทั้งหมดที่ต้องการได้แล้วจะทำการเก็บข้อมูลลงในระบบฐานข้อมูลซึ่งมีรายละเอียดดังนี้

TABLE

ตารางที่ 4.2 แสดงรายละเอียดของ Table TBPrint

ชื่อตาราง	ชื่อ field	ชื่อตัวแปร	Description
TBPrint	UserId	Text	รหัสของผู้ใช้
	TimePrinted	Date/Time	เวลาที่มีการพิมพ์
	DatePrinted	Date/Time	วันเดือนปีที่พิมพ์
	JobName	Text	ชื่อของ job ที่พิมพ์
	SizePrinted	Integer	ขนาดที่มีการพิมพ์
	TotalPg	Integer	จำนวนแผ่นของกระดาษ

ตารางที่ 4.3 แสดงรายละเอียดของ Table TBuser

ชื่อตาราง	ชื่อ field	ชื่อตัวแปร	Description
TBuser	UserID	Text	รหัสของผู้ใช้
	UserName	Text	รายชื่อผู้ใช้
	Pgused	Integer	จำนวนแผ่นกระดาษรวม
	LimitPg	Integer	จำกัดจำนวนแผ่นกระดาษ

4.6 การประมาณค่าขนาดของข้อมูล ต่อ 1 แผ่นกระดาษ

เนื่องจากการส่งข้อมูลผ่านระบบ Network จะทำการส่งในรูปแบบของ Byte ทำให้โปรแกรมไม่สามารถเก็บข้อมูลจำนวนแผ่นของเอกสารได้ ดังนั้นจึงต้องนำขนาดของข้อมูลที่โปรแกรมเก็บได้มาทำการคำนวณให้เป็นจำนวนแผ่น

หลักการประมาณค่าขนาดของข้อมูล ต่อ 1 แผ่นกระดาษ จะต้องมีการสุ่มตัวอย่างเอกสารขึ้นมาจำนวนหนึ่ง แล้วนำขนาดข้อมูลของแต่ละเอกสารที่โปรแกรมสามารถเก็บได้มารวมกัน เป็นค่า $\sum x$ และนำจำนวนแผ่นกระดาษทั้งหมดของแต่ละเอกสารมารวมกัน เป็นค่า n หลังจากนั้นทำการหาค่าเฉลี่ยของขนาดข้อมูล ต่อ 1 แผ่นกระดาษ จากสูตร $\overline{x} = \sum x/n$

การสุ่มตัวอย่างเอกสารขึ้นมาจำนวนหนึ่งได้รายละเอียดดังนี้

ตารางที่ 4.4 แสดงรายละเอียดของการส่มตัวอย่างเอกสาร

ชนิดของซอฟท์แวร์	ขนาดไฟล์ที่สั่งพิมพ์	จำนวนแผ่นที่สั่งพิมพ์	จำนวนแผ่นที่ประมาณ
Word	1.98 MB	18	13
Word	90.8 KB	1	1
Word	416 KB	2	3
Word	4.81 MB	29	31
Word	101 KB	1	1
Word	326 KB	1	2
Word	353 KB	2	3
Word	456 KB	2	3
Word	421 KB	2	3
Notepad	45.6 KB	1	1
Notepad	312 KB	2	2
Notepad	436 KB	3	3
Notepad	676 KB	3	5
Notepad	187 KB	2	2
Notepad	15.3 KB	5	6
Notepad	837 KB	3	4
Notepad	568 KB	2	2
Notepad	285 KB	5	7
Notepad	0.99 MB	2	2
Notepad	296 KB	7	7
Notepad	1.12 KB	4	5
Notepad	251 KB	1	2
Notepad	117 KB	1	1
Notepad	130 KB	1	1
Internet Explorer	449 KB	3	3
Internet Explorer	492 KB	3	3

ตารางที่ 4.4(ต่อ) แสดงรายละเอียดของการสุ่มตัวอย่างเอกสาร

Internet Explorer	91.9 KB	1	1
Internet Explorer	229 KB	2	2
Internet Explorer	586 KB	3	4
Internet Explorer	112 KB	2	1

จากตารางข้างบนสามารถคำนวณหาขนาดข้อมูลของเอกสารทั้งหมด ซึ่งจะได้ขนาดเท่ากับ 20488.62 KB และสามารถคำนวณหาแผ่นกระดาษของเอกสารทั้งหมดได้เท่ากับ 125 แผ่น ดังนั้นจึงสามารถคำนวณได้ว่ากระดาษ 1 แผ่นมีขนาด 163.9 KB แต่จะใช้ค่าประมาณเท่ากับ 164 KB

บทที่ 5
สรุปผลการศึกษาและข้อเสนอแนะ

5.1 สรุปผลปัญหาพิเศษ

ในการศึกษาและสร้างโปรแกรมจัดการการใช้งานของเครื่องพิมพ์ได้มีวัตถุประสงค์ที่จะทำการนับจำนวนแผ่นกระดาษที่นักศึกษาแต่ละพิมพ์ได้ ซึ่งในระบบเครือข่ายเมื่อมีการสั่งพิมพ์เอกสารจากเครื่อง client ไปยัง Print Server มีการส่งขนาดไฟล์ ซึ่งจะต้องทำการประมาณจำนวนแผ่นกระดาษในการสั่งพิมพ์ในแต่ละไฟล์ ซึ่งมีการเก็บรวบรวมข้อมูลอยู่ในบทที่ 4 ซึ่งได้ว่ากระดาษ 1 แผ่นมีขนาด 164 KB ดังนั้นในการนับกระดาษอาจจะไม่ตรงกับความเป็นจริงในบางครั้ง

5.2 ข้อเสนอแนะ

5.2.1 จากการศึกษาปัญหาพิเศษหัวข้อเรื่องโปรแกรมจัดการการใช้งานเครื่องพิมพ์พบว่า การส่งข้อมูลจากเครื่อง client ไปยัง Print Server ในระบบเครือข่ายจะมีการส่งขนาดของไฟล์เป็นไบต์ ซึ่งทำให้ไม่สามารถนับจำนวนแผ่นที่ได้จากการพิมพ์ได้อย่างถูกต้อง จึงควรที่จะมำการควบคุมในส่วนของพอร์ตของเครื่องพิมพ์จะทำให้ได้ข้อมูลที่ถูกต้องมากที่สุด

5.2.2 ควรใช้ระบบปฏิบัติการวินโดวส์เอ็นทีบน Print Server เพื่อให้การตรวจสอบ username และ password ของนักศึกษาที่ทำการสั่งพิมพ์มีความปลอดภัยมากยิ่งขึ้น

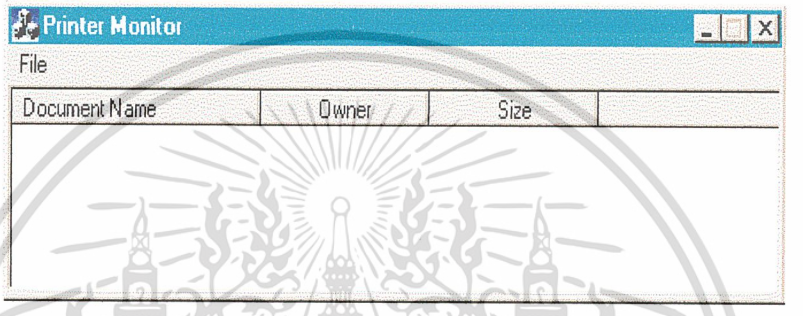
ภาคผนวก ก.
คู่มือการใช้โปรแกรม

วิธีใช้โปรแกรม Print Controller

โปรแกรมควบคุมการทำงานของเครื่องพิมพ์ได้แบ่งการทำงานออกเป็น 2 ส่วนหลักๆ คือ

1. **ส่วนของการนับแผ่นกระดาษ**

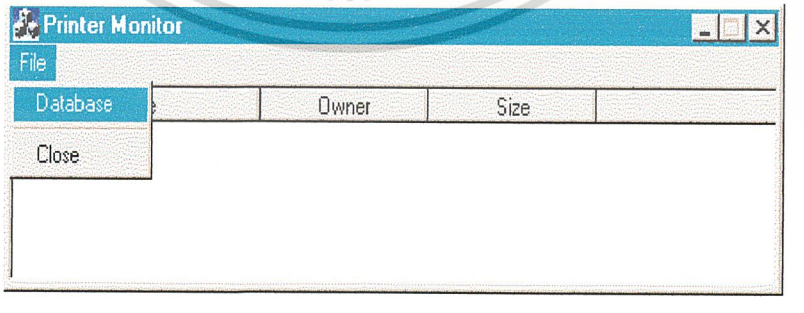
1. เริ่มต้น Run โปรแกรมจะมีหน้าจอขึ้นดังนี้



รูปที่ ก-1 แสดงโปรแกรมขณะ Run

สถานะข้างล่างจะเป็น Running แสดงว่าขณะนี้โปรแกรมพร้อมที่จะทำการตรวจสอบงานพิมพ์ ส่วนใน Control list จะทำการแสดงรายละเอียดเกี่ยวกับงานพิมพ์ ณ ขณะนั้น

2. การดูข้อมูลที่ถูกเก็บไว้ในระบบฐานข้อมูลให้คลิกไปที่ Menu File แล้วเลือก Database ดังรูปที่ ก-2

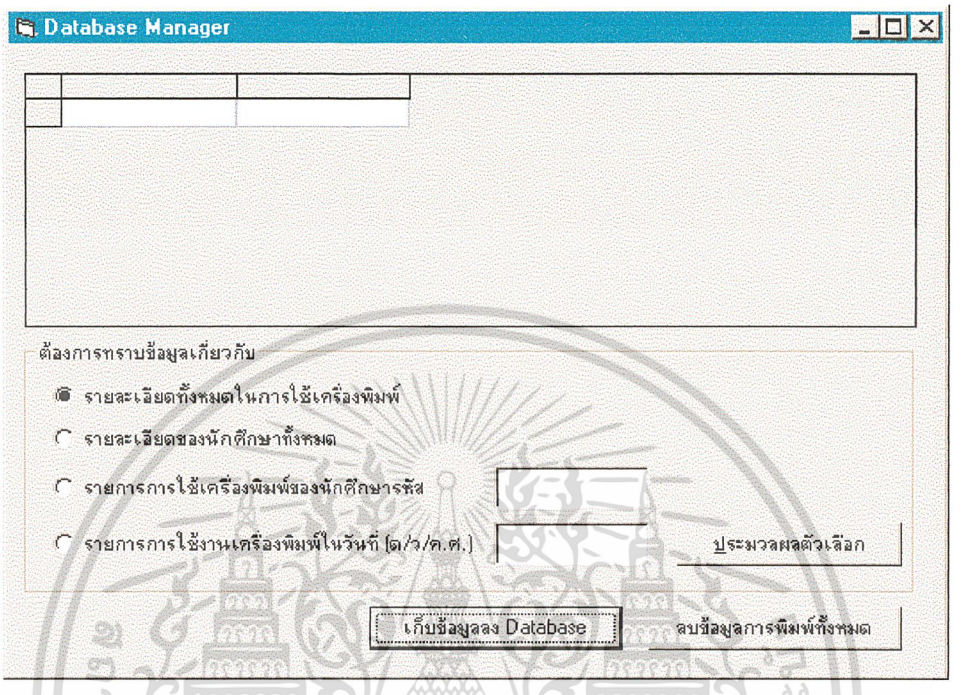


รูปที่ ก-2 การเข้าดู Database

3. เมื่อต้องการจบโปรแกรมให้คลิกที่ Close หรือคลิกที่ปุ่ม X ที่มุมบนขวาของโปรแกรม

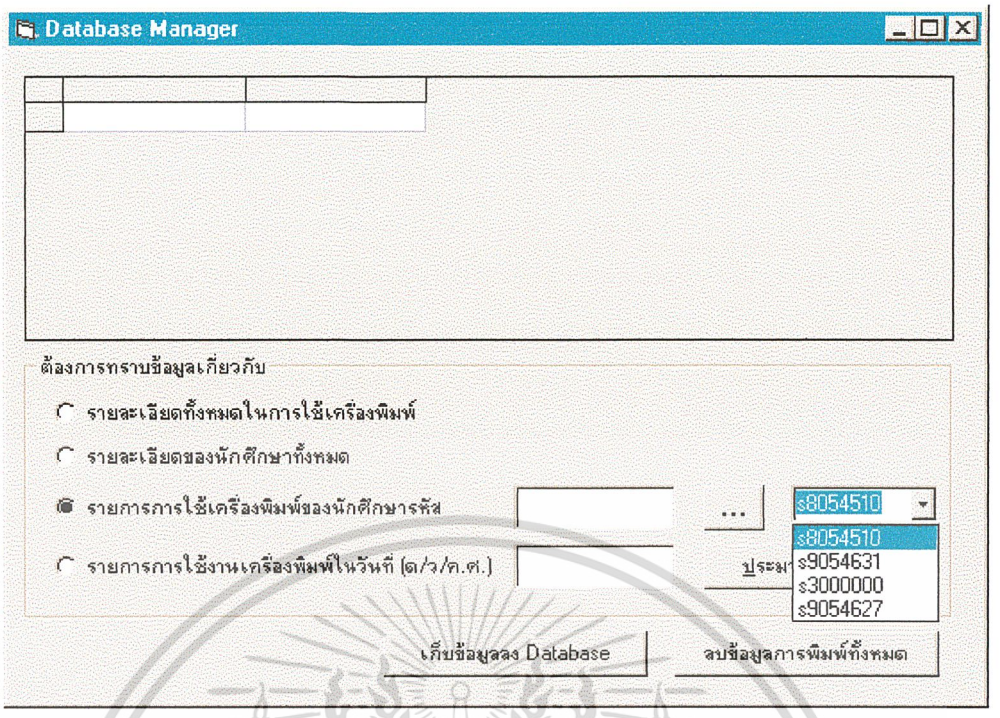
2. ส่วน Database

1. เมื่อเริ่มการทำงานในส่วนนี้จะปรากฏหน้าจอดังรูป



รูปที่ ก-3 แสดงการเริ่มใช้ Database

2. ถ้าต้องการนำข้อมูลการสั่งพิมพ์ที่เก็บอยู่ใน File ที่ถูกสร้างโดยโปรแกรม มาเก็บใน Database ให้กดปุ่ม เก็บข้อมูลลง Database
3. ถ้าต้องการทราบข้อมูลการพิมพ์ของนักศึกษาทั้งหมด ให้เลือกตัวเลือกที่ 1
4. ถ้าต้องการทราบข้อมูลเกี่ยวกับนักศึกษา ให้เลือกตัวเลือกที่ 2
5. ถ้าต้องการทราบข้อมูลการพิมพ์ของนักศึกษาเฉพาะคน ให้เลือกตัวเลือกที่ 3 ซึ่งจะต้องระบุรหัสนักศึกษา แต่ถ้าไม่ทราบรหัสนักศึกษาสามารถทำการค้นหาได้โดยการกดปุ่ม เพื่อเลือกรหัสนักศึกษา แล้วทำการเลือกรหัสนักศึกษาที่ต้องการทราบข้อมูล จะพบว่ารหัสนักศึกษาที่ถูกเลือกจะไปปรากฏในช่องสำหรับใส่รหัสนักศึกษา ดังรูปที่ ก-4



รูปที่ ก-4 แสดงการเลือกรหัสนักศึกษา

6. ถ้าต้องการทราบข้อมูลในแต่ละวัน ให้เลือกที่ตัวเลือกที่ 4 โดยให้ใส่ข้อมูลวันที่ต้องการจะทราบ ลงในช่องวันที่โดยใส่ตามลำดับ เดือน/วัน/ปี ค.ศ.
7. หลังจากทำการเลือกตัวเลือกเรียบร้อยแล้วให้คลิกที่ปุ่ม ประมวลตัวเลือก จะปรากฏข้อมูลตามที่เลือกในตารางบนตัวเลือก
8. ถ้าต้องการลบข้อมูลเกี่ยวกับการพิมพ์ของนักศึกษาทุกคน ให้คลิกที่ปุ่ม ลบข้อมูลการพิมพ์ทั้งหมด
9. เมื่อต้องการออกจากส่วน Database ให้คลิกที่ปุ่ม x ที่มุมบนขวาของโปรแกรม

ภาคผนวก ข.

Source Code

```
#include "stdafx.h"
#include "TestFinal.h"
#include "TestFinalDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA


        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
```

```
protected:
        //{{AFX_MSG(CAboutDlg)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////
```

```
// CTestFinalDlg dialog

CTestFinalDlg::CTestFinalDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CTestFinalDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CTestFinalDlg)
        //}}AFX_DATA_INIT
        // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CTestFinalDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CTestFinalDlg)
        DDX_Control(pDX, IDC_LISTCONTROL, m_clist);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTestFinalDlg, CDialog)
        //{{AFX_MSG_MAP(CTestFinalDlg)
        ON_WM_SYSCOMMAND()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
        ON_COMMAND(ID_FILE_DATABASE, OnFileDatabase)
        ON_COMMAND(ID_FILE_CLOSE, OnFileClose)
        ON_WM_SPOOLERSTATUS()
        ON_WM_TIMER()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////////////////////////////////////////////
// CTestFinalDlg message handlers

BOOL CTestFinalDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon
```

```
// TODO: Add extra initialization here

column.mask =LVCF_FMT|LVCF_SUBITEM|LVCF_TEXT|LVCF_WIDTH;
column.fmt = LVCFMT_CENTER;

column.pszText = "Document Name";
column.iSubItem = 0;
column.cx = 150;
m_clist.InsertColumn(0,&column);

column.pszText = "Owner";
column.iSubItem = 1;
column.cx = 100;
m_clist.InsertColumn(1,&column);

column.pszText = "Size";
column.iSubItem = 2;
column.cx = 100;
m_clist.InsertColumn(2,&column);

PrinterOpen();
ListJob();
if( n != 0)
{
        UpdateItem(pJobInfo,n);
}
else
{
        free(pJobInfo);
}
id = 0;
```

```
        MakeFile();

        return TRUE;  // return TRUE  unless you set the focus to a control
}

void CTestFinalDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CTestFinalDlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this); // device context for painting

                SendMessage(WM_ICONERASEBKGND, (WPARAM)   dc.GetSafeHdc(),
                        0);
```

```
                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CTestFinalDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}
```

เมื่อมีการคลิกที่ปุ่ม Database ใน Menu จะให้ไปเรียกไฟล์ Database ขึ้นมาโดยใช้ฟังก์ชันดังต่อไปนี้

```
void CTestFinalDlg::OnFileDatabase()
{
        // TODO: Add your command handler code here
        WinExec("c:/DBPrinter/prjPrinter.exe",SW_SHOW);
}
```

เมื่อมีการคลิกที่ปุ่ม Close ใน Menu จะทำการปิดโปรแกรม โดยใช้ ฟังก์ชันดังต่อไปนี้

```
void CTestFinalDlg::OnFileClose()
{
        // TODO: Add your command handler code here
        ClosePrinter(hPrinter);
        MessageBox("Close Printer");
        PostQuitMessage(0);
}

void CTestFinalDlg::PrinterOpen()
{
        int i = 0;
        char temp_str[80] = "";
        GetProfileString("windows","device",",",buff,sizeof(buff));

        while (buff[i] != ',')
                        temp_str[i++] = buff[i];

        pPrinterName = temp_str;

        CString Printername;
        Printername = pPrinterName;

        if(Printername == "HP DeskJet 850C")
        {
```

```
            sprintf(buff,Printername);
            MessageBox(buff);
            pagenum = 335876;
        }
        else
        {
            sprintf(buff,Printername);
            MessageBox(buff);
            pagenum = 167936;
        }

        if(!OpenPrinter(pPrinterName,&hPrinter,NULL))
        {
            MessageBox("Open Printer Error");
            DestroyWindow();
        }
        else
        {
            MessageBox("Open Printer");
        }

        number = 0;
}
```

ฟังก์ชัน PrinterOpen จะทำหน้าที่ Set ค่าเริ่มต้นต่างๆที่จำเป็นในการเริ่มทำงานกับ Printer โดยเริ่มต้นด้วยการ get ชื่อของ Printer มาจาก buffer และทำการตรวจสอบว่าเป็น Printer รุ่นใด เพื่อใช้ในการประมาณขนาดของเอกสารหนึ่งแผ่น เมื่อทำการตรวจสอบได้แล้วจึงใช้ฟังก์ชัน OpenPrinter เพื่อหาค่า &hPrinter เพื่อใช้เป็นตัวจัดการการทำงานของ Printer

```
BOOL CTestFinalDlg::DestroyWindow()
{
        // TODO: Add your specialized code here and/or call the base class
        ClosePrinter(hPrinter);
```

```
        MessageBox("Close Printer");
        StopTime(timevalue);
        return CDialog::DestroyWindow();
}
```

ฟังก์ชัน DestroyWindow เป็นการปิด window ที่ทำงานอยู่

```
void CTestFinalDlg::OnSpoolerStatus(UINT nStatus, UINT nJobs)
{
        CDialog::OnSpoolerStatus(nStatus, nJobs);

        // TODO: Add your message handler code here
                if (number < nJobs)
        {
                number = nJobs;
                ListJob();

                if (number == 1)        // First job
                {
                        JobAdd();
                }
                else                    // Other job
                {
                        OtherJobAdd();
                }

        }

        else
        {
                number = nJobs;
                StopTime(timevalue);
```

```
            RemoveJob();
        }
}
```

ฟังก์ชันนี้เป็นการดักเมสเสจ WM_SPOOLERSTATUS ซึ่งจะเกิดขึ้นเมื่อมีการสั่งพิมพ์งานจากเครื่องใดๆ บนระบบเครือข่าย ซึ่งเมื่อมีเมสเสจเข้ามาจะทำการตรวจสอบว่าเป็นเมสเสจที่เกิดขึ้น เมื่อมีการสั่งพิมพ์หรือเกิดขึ้นเมื่อมีงานพิมพ์พิมพ์สำเร็จแล้ว ถ้ามีการสั่งพิมพ์โดยเข้ามาเป็นงานแรกจะให้ไปทำฟังก์ชัน JobAdd(); แต่ถ้าไม่ใช่งานแรก ให้ไปทำฟังก์ชัน OtherJobAdd(); และถ้าเป็นเมสเสจที่เกิดขึ้นเมื่อการพิมพ์งานเสร็จให้ไปทำฟังก์ชัน RemoveJob();

```
void CTestFinalDlg::CheckPrintJob()
{
        int i;
        int p_id = 0;
        bool p;

        p = false;

        for(i = 0;i<=n;i++)
        {
            if(pJobInfo[i].Status & JOB_STATUS_PRINTING)
            {
                p_id = pJobInfo[i].JobId;
        //      sprintf(buff,"CheckPrintJob id = %d",p_id);
                p = true;
            }
        }
        if(p)
        {
            printid = p_id;
        }
        else
```

```
        {
                printid = -1;
        }
}
```

ฟังก์ชัน CheckPrintJob(); ทำหน้าที่หาว่าในขณะนี้ เครื่องกำลังทำการพิมพ์งานที่เท่าไรอยู่

```
int CTestFinalDlg::FindJob(int findid)
{
        int i = 0;
        bool loop = true;
        do
        {
          if(i > n)
          {
                loop = false;
          }
          else
          {
                if(pJobInfo[i].JobId == (DWORD)findid)
                {
                        loop = false;
                        return(i);
                }
          }

          i = i+1;
        }while(loop);
        return(0);
}
```

ฟังก์ชัน FindJob(); ทำการตรวจสอบว่า งานพิมพ์ที่เราต้องการทราบอยู่ในตำแหน่ง array ที่เท่าไรของ pJobInfo

```
void CTestFinalDlg::FindPrintJob()
{
	int i;
	int p_id = 0;
	bool p;
	p = false;
	for(i = 0;i<=nout;i++)
	{
		if(pJobInfo2[i].Status & JOB_STATUS_PRINTING)
		{
			p_id = pJobInfo2[i].JobId;
			sprintf(buff,"FindPrintid = %d",p_id);
			p = true;
		}
	}
	if(p)
	{
		printid = p_id;
	}
	else
	{
		printid = -1;
	}
}
```

ฟังก์ชัน FindPrintJob(); ทำการตรวจสอบว่า งานพิมพ์ที่เราต้องการทราบอยู่ในตำแหน่ง arrayที่เท่าไรของ pJobInfo2


```
void CTestFinalDlg::GetTime()
```

```
{
        _strdate(Date);
        _strtime(Time);
}
```

ฟังก์ชัน GetTime(); ทำการ get ค่าวันและเวลามาจาก Window

```
void CTestFinalDlg::JobAdd()
{
        id = pJobInfo[n-1].JobId;
        printid = pJobInfo[n-1].JobId;
        LoopCheck();
}
```

ฟังก์ชัน JobAdd() จะทำการหาค่า id ของงานพิมพ์แล้วทำการวนลูปตรวจสอบโดยใช้ฟังก์ชัน LoopCheck(); เพื่อทำการตรวจสอบว่างานพิมพ์นั้นมีการเปลี่ยนแปลงใดๆ หรือไม่

```
void CTestFinalDlg::ListJob()
{
        DWORD dwNeeded,dwReturned;
        if(!EnumJobs(hPrinter,0,0xFFFFFFFF,2,NULL,0,&dwNeeded, &dwReturned))
        {
                if(GetLastError() != ERROR_INSUFFICIENT_BUFFER)
                {
                        ClosePrinter(hPrinter);
                        MessageBox("Enum Error");
                }
        }

        if ((pJobInfo = (JOB_INFO_2 *)malloc(100000)) == NULL)
        {
                ClosePrinter(hPrinter);
                MessageBox("Job Info 2 Error");
        }
```

```
        if (!EnumJobs(hPrinter,0,0xFFFFFFFF,2,(LPBYTE)pJobInfo,dwNeeded,
          &dwNeeded,&dwReturned))
        {
                ClosePrinter(hPrinter);
                MessageBox("EnumJobs Error");
        }

        n = dwReturned;
        if(n == 0)
        {
                m_clist.DeleteAllItems();
        }
        else
        {
                UpdateItem(pJobInfo,n);
        }
}
```

ฟังก์ชัน ListJob(); จะทำการเก็บงานทั้งหมดที่อยู่ Spooler ลงใน array pJobInfo

```
void CTestFinalDlg::KeepData(int keepid)
{
        int pages,sizes,pSize;
        GetTime();
        sizes = pJobInfo[keepid].Size;
        pages = sizes / pagenum;
        pSize = sizes % pagenum;
        if(pSize != 0)
        {
                pages = pages + 1;
        }
```

```
        sprintf(buff,"%s&&%s&&%d&&%d&&%s&&%s&&",pJobInfo[keepid].pUserName,
                pJobInfo[keepid].pDocument,pages,pJobInfo[keepid].Size,Date,Time);
        ListFile(buff);
}
```

ฟังก์ชัน KeepData(); จะทำหน้าที่เก็บข้อมูลที่จำเป็นคือ ชื่อผู้สั่งพิมพ์ ชื่องานที่พิมพ์ จำนวนหน้ากระดาษ ขนาดของงานพิมพ์ วัน และเวลาลงใน File

```
void CTestFinalDlg::OtherJobAdd()
{
        id = pJobInfo[n-1].JobId;
        CheckPrintJob();
        if(printid == -1)
        {
                printid = id;
        }
        if(id == printid)
        {
                LoopCheck();
        }
}
```

```
UINT CTestFinalDlg::TimeSet()
{
        return SetTimer(1,5000,0);
}
```

ฟังก์ชัน TimeSet() เป็นการกำหนดเวลา 5 วินาที ในการวนลูปแต่ละครั้ง

```
void CTestFinalDlg::StopTime(UINT stop)
{
```

```
        KillTimer(stop);
}
```

ฟังก์ชัน StopTime() เป็นการกำหนดให้หยุดการวนลูป

```
void CTestFinalDlg::OnTimer(UINT nIDEvent)
{
        // TODO: Add your message handler code here and/or call default
        free(pJobInfo);
        ListJob();
        UpdateItem(pJobInfo,n);
        CheckPrintJob();
        CDialog::OnTimer(nIDEvent);
}
```

ฟังก์ชัน OnTimer() เป็นการบอกว่าในการวนลูปแต่ละครั้งจะต้องทำการตรวจสอบว่ามีความเปลี่ยนแปลงหรือไม่โดยใช้ ฟังก์ชัน CheckPrintJob()

```
void CTestFinalDlg::LoopCheck()
{
        timevalue = TimeSet();
}
```

ฟังก์ชัน LoopCheck() เป็นการกำหนดว่าในการวนลูปแต่ละครั้งจะต้องวนทุกๆ เวลาที่กำหนดไว้คือเวลา 5 วินาที

```
void CTestFinalDlg::RemoveJob()
{
        ListOutJob();

        if(number == 0)
        {
                if(printid >= 0)
                {
```

```
                countid = FindJob(printid);
                KeepData(countid);
            }
        }
        else
        {
            if(FindJob2(printid))
            {
                LoopCheck();
            }
            else
            {
                if(printid >= 0)
                {
                    countid = FindJob(printid);
                    KeepData(countid);
                }
                FindPrintJob();
                free(pJobInfo2);
                LoopCheck();
            }
        }
}
void CTestFinalDlg::ListOutJob()
{
    DWORD dwNeeded,dwReturned;

    if(!EnumJobs(hPrinter,0,0xFFFFFFFF,2,NULL,0,&dwNeeded, &dwReturned))
    {
        if(GetLastError() != ERROR_INSUFFICIENT_BUFFER)
        {
```

```
                ClosePrinter(hPrinter);
                MessageBox("Enum Error");
            }
        }

        if ((pJobInfo2 = (JOB_INFO_2 *)malloc(100000)) == NULL)
        {
            ClosePrinter(hPrinter);
            MessageBox("Job Info 2 Error");
        }

        if (!EnumJobs(hPrinter,0,0xFFFFFFFF,2,(LPBYTE)pJobInfo2,dwNeeded,
            &dwNeeded,&dwReturned))
        {
            ClosePrinter(hPrinter);
            MessageBox("EnumJobs Error");
        }

        nout = dwReturned;


        if(nout == 0)
        {
            m_clist.DeleteAllItems();
        }
        else
        {
            UpdateItem(pJobInfo2,nout);
        }
    }
```

ฟังก์ชัน ListJob(); จะทำการเก็บงานทั้งหมดที่อยู่ Spooler ลงใน array pJobInfo2

```
void CTestFinalDlg::ListFile(char *idnum)
{
        CStdioFile file("c:\\DBPrinter\\Test.txt",CFile::modeWrite);
        file.SeekToEnd();
        file.WriteString(idnum);
        file.WriteString("\n");
}
```

ฟังก์ชัน ListFile() จะทำการเก็บข้อมูลลง File ชื่อ “Test.txt “

```
void CTestFinalDlg::MakeFile()
{
        CStdioFile files("c:\\DBPrinter\\Test.txt",CFile::modeCreate);
        files.WriteString("\n");
}
```

ฟังก์ชัน MakeFile() จะทำการสร้าง File ชื่อ Test.txt ไว้ใน directory ชื่อ DBPrinter

```
bool CTestFinalDlg::FindJob2(int findid)
{
        int i = 0;
        bool loop = true;

        do
        {
         if(i > nout)
         {
                loop = false;
         }
         else
```

```
        {
                if(pJobInfo2[i].JobId == (DWORD)findid)
                {
                        loop = false;
                        return(true);
                }
        }

        i = i+1;
    }while(loop);

    return(false);
}

void CTestFinalDlg::UpdateItem(JOB_INFO_2 *pItem, int countitem)
{
    int i;
    m_clist.DeleteAllItems();

    for(int j = 0;j < countitem;j++)
    {
            item.mask = LVIF_TEXT;
            item.iItem = j;
            item.iSubItem = 0;
            item.pszText = pItem[j].pDocument;
            i = m_clist.InsertItem(&item);

            item.mask = LVIF_TEXT;
            item.iItem = i;
            item.iSubItem = 1;
            item.pszText = pItem[j].pUserName;
```

```
                m_clist.SetItem(&item);

                sprintf(buff,"%d",pItem[j].Size);
                item.iSubItem = 2;
                item.pszText = buff;
                m_clist.SetItem(&item);
        }
}
```

ฟังก์ชัน UpdateItem() จะทำหน้าที่คอยเปลี่ยนแปลงหน้าจอ ให้แสดงงานทั้งหมดที่อยู่ใน Spooler ในขณะปัจจุบัน

บรรณานุกรม

นิรุธ อำนวยศิลป์. 2542. **Visual C++ Version 6.0.** กรุงเทพมหานคร : บริษัท ส.เอเชียเพรส (1989) จำกัด

วิภา เพิ่มทรัพย์ และสิดารัศมิ์ ธำรงสมบัติสกุล. 2541. **จับประเด็น Microsoft Windows 98.** กรุงเทพมหานคร : บริษัท เอช. เอ็น. กรุ๊ป จำกัด

กิตติ ภักดีวัฒนะกุล และจำลอง ครูอุตสาหะ. 2542. **Visual Basic 6 ฉบับโปรแกรมเมอร์.** กรุงเทพมหานคร : ไทยเจริญการพิมพ์