โปรแกรมควบคุมการใช้คอมพิวเตอร์ในห้องปฏิบัติการ

PC-LABORATORY CONTROL PROGRAM

คเชนทร์ สิทธิโชค
พีรศักดิ์ วิวัฒรางกูล
รัศมิมาน หาวนพันธ์
สายฝน เขาหิน

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2542

PC-LABORATORY CONTROL PROGRAM

KACHANE SITTHICHOKE

PEERASAK WIWATTARANGKUL

RATSAMIMARN HAVANAPHAN

SAIPHON KHAOHIN

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

ACADEMIC YEAR 1999

หัวข้อปัญหาพิเศษ	โปรแกรมควบคุมการใช้คอมพิวเตอร์ในห้องปฏิบัติการ	
	PC-LABORATORY CONTROL PROGRAM	
ชื่อนักศึกษา	นายคเชนทร์ สิทธิโชค	39054609
	นายพีรศักดิ์ วิวัฒรางกูล	39054647
	นางสาวรัศมิมาน หาวนพันธ์	39054655
	นางสาวสายฝน เขาหิน	39054671
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	
สาขาวิชา	วิทยาการคอมพิวเตอร์	
ปีการศึกษา	2542	
อาจารย์ที่ปรึกษา	อาจารย์ ไพรบูลย์ พันธรักษ์พงษ์	

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์ ประจำปีการศึกษา 2542

คณะกรรมการสอบ		ลายมือชื่อ
ประธานกรรมการ	อาจารย์วีระชัย ตันยะสิทธิ์	
กรรมการ	อาจารย์นันทิกา เบญจเทพานันท์	
กรรมการและอาจารย์ที่ปรึกษา	อาจารย์ไพรบูลย์ พันธรักษ์พงษ์	

(อาจารย์ไพรบูลย์ พันธรักษ์พงษ์)
หัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

หัวข้อปัญหาพิเศษ	โปรแกรมควบคุมการใช้คอมพิวเตอร์ในห้องปฏิบัติการ
ชื่อนักศึกษา	นายคเชนทร์ สิทธิโชค 39054609
	นายพีรศักดิ์ วิวัฒเรางกูล 39054647
	นางสาวรัศมิมาน หาวนพันธ์ 39054655
	นางสาวสายฝน เขาหิน 39054671
ปริญญา	วิทยาศาสตรบัณฑิต
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สาขาวิชา	วิทยาการคอมพิวเตอร์
ปีการศึกษา	2542
อาจารย์ที่ปรึกษา	อาจารย์ไพรบูลย์ พันธรักษ์พงษ์

บทคัดย่อ

ในปัจจุบันนี้คอมพิวเตอร์ได้เข้ามามีบทบาทและความสำคัญต่อการทำงานของมนุษย์ในทุกระบบงาน ทั้งด้านธุรกิจ ด้านการสื่อสารและด้านการศึกษา เป็นต้น การใช้งานคอมพิวเตอร์ให้ได้รับประโยชน์สูงสุดนั้น จำเป็นอย่างยิ่งที่จะต้องมีเครื่องมือสนับสนุนที่ดีและทรัพยากรต่าง ๆ เพียงพอ ทรัพยากรมนุษย์จัดเป็นทรัพยากรหนึ่งที่มีความสำคัญต่อหน่วยงาน นอกจากนี้ยังเป็นทรัพยากรที่สามารถพัฒนาความสามารถของตนเองได้ และเพื่อให้ทรัพยากรมนุษย์มีความรู้ความสามารถและความพร้อมในการที่จะใช้งานคอมพิวเตอร์ได้อย่างมีประสิทธิภาพ การพัฒนาทรัพยากรมนุษย์ในด้านการเรียนการสอนคอมพิวเตอร์จึงมีความสำคัญและต้องกระทำอย่างมีคุณภาพตามแนวทางที่ถูกต้อง บุคลากรผู้ให้ความรู้ต้องเป็นผู้มีความรู้ความสามารถ มีอุปกรณ์คอมพิวเตอร์ที่ได้มาตรฐานและพอเพียง มีห้องปฏิบัติการคอมพิวเตอร์ให้ใช้ศึกษาหาความรู้ได้อย่างเต็มที่

ดังนั้นระบบการควบคุมห้องปฏิบัติการคอมพิวเตอร์ จึงมีความจำเป็นอย่างยิ่งเพื่อให้เกิดระเบียบภายในห้องปฏิบัติการ เพื่อให้นักศึกษาที่เข้ามาใช้งานคอมพิวเตอร์ในการปฏิบัติงานด้านต่าง ๆ เช่น เขียนโปรแกรม ค้นหาข้อมูลทางอินเตอร์เน็ต เป็นต้น สามารถใช้งานได้อย่างเต็มที่ อีกทั้งยังต้องอำนวยความสะดวกให้แก่ผู้ควบคุมห้องปฏิบัติการคอมพิวเตอร์ เพื่อให้สามารถควบคุมดูแลการใช้งานห้องปฏิบัติการคอมพิวเตอร์ได้อย่างมีประสิทธิภาพมากขึ้น

Special Project Title	PC-Laboratory Control Program	
Students	Mr. Kachane Sitthichoke	39054609
	Mr. Peerasak Wiwattarangkul	39054647
	Miss Ratsamimarn Havanaphan	39054655
	Miss Saiphon Khaohin	39054671
Degree	Bachelor's Degree of Science	
Department	Mathematics and Computer Sciences , Faculty of Science	
Programme	Computer Sciences	
Academic Year	1999	
Special Project Advisor	Lecturer Praiboon Pantaragphong	

ABSTRACT

In the present days , computer has more involved with our lives in many ways such as in Business , Communication and Education. To get the most useful in using computer , we need many good tools and resources to support in. One of many resources that important is human resources. Because human resources can develop their abilities and skills all the time so to develop human resources to have more computer skills is important and have to do in the right way by concern with the quality too.

To do this we need computer laboratory to learn and find out more computer knowledges and skills. So it is necessary that we have to find the way to control computer laboratory effectively.

กิตติกรรมประกาศ

ในการทำปัญหาพิเศษเรื่องโปรแกรมควบคุมการใช้คอมพิวเตอร์ในห้องปฏิบัติการสามารถสำเร็จลุล่วงไปด้วยดี คณะผู้จัดทำต้องขอขอบพระคุณ อาจารย์ไพรบูลย์ พันธรักษ์พงษ์ อาจารย์ผู้รับผิดชอบปัญหาพิเศษฉบับนี้ที่กรุณาให้คำแนะนำ และเป็นที่ปรึกษาในการแก้ปัญหาต่าง ๆ รวมทั้งเป็นผู้ตรวจสอบความถูกต้องของปัญหาพิเศษฉบับนี้

นอกจากนี้คณะผู้จัดทำต้องขอขอบพระคุณ บิดา มารดา ที่ได้ให้ความสนับสนุนทางด้านกำลังใจและทุนทรัพย์ จนการทำปัญหาพิเศษครั้งนี้สำเร็จด้วยดี รวมทั้งเพื่อน ๆ และน้อง ๆ ทุกคนที่ให้ความช่วยเหลือในด้านต่าง ๆ เกี่ยวกับปัญหาพิเศษไว้ ณ ที่นี้

คณะผู้จัดทำ

มีนาคม 2543

สารบัญ

หน้า

สารบัญตาราง

สารบัญภาพ

บทที่1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหาพิเศษ

เนื่องจากการเรียนการสอนทางคอมพิวเตอร์ของภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ จำเป็นอย่างยิ่งที่จะต้องใช้คอมพิวเตอร์ในการปฏิบัติงานด้านต่าง ๆ เช่น เขียนโปรแกรม ค้นหาข้อมูลทางอินเตอร์เน็ต ดังนั้นทางภาควิชาจึงได้จัดให้มีห้องปฏิบัติการคอมพิวเตอร์ขึ้น เพื่อให้นักศึกษาได้เข้ามาใช้งานและได้จัดให้มีผู้ควบคุมดูแลภายในห้องปฏิบัติการซึ่งทำหน้าที่ควบคุมดูแลให้นักศึกษาที่เข้ามาใช้ห้องปฏิบัติการดำเนินงานตามกฎระเบียบที่ได้กำหนดไว้ แต่ก็ยังมีนักศึกษาจำนวนมากที่มักจะละเมิดกฎ เช่น เปิดดูเว็บไซต์ที่ไม่ได้รับอนุญาต เล่นเกม ฯลฯ ทำให้นักศึกษาที่ต้องการใช้ห้องปฏิบัติการจริง ๆ ไม่มีเครื่องคอมพิวเตอร์ไว้ใช้ นักศึกษาเหล่านั้นจึงไม่ได้รับความสะดวกเท่าที่ควร นอกจากนี้ผู้ดูแลระบบซึ่งมีเพียงคนเดียวยังมีหน้าที่อื่น ๆ ที่ต้องปฏิบัติอีก เช่น การดูแลรักษาระบบต่าง ๆ ในห้องปฏิบัติการ ทำให้การควบคุมดูแลการใช้งานห้องปฏิบัติการคอมพิวเตอร์ให้ถูกต้องตามกฎระเบียบเป็นไปได้อย่างยากลำบาก เนื่องจากผู้ดูแลระบบจะต้องคอยเดินมาดูที่หน้าจอของแต่ละเครื่องจึงจะรู้ว่านักศึกษาทำผิดกฎหรือไม่ ซึ่งอาจทำให้นักศึกษารู้ตัวก่อนแล้วทำการปิดแอพพลิเคชันนั้นเสียทำให้ไม่สามารถเอาผิดได้ นอกจากนี้ห้องปฏิบัติการคอมพิวเตอร์ยังมีขนาดใหญ่และมีเครื่องคอมพิวเตอร์ลูกข่ายเป็นจำนวนมาก

ด้วยเหตุผลข้างต้นจึงทำให้ทีมงานเกิดความคิดที่จะพัฒนาโปรแกรมเพื่อแก้ไขปัญหาดังกล่าวและอำนวยความสะดวกในการควบคุมดูแลการใช้งานห้องปฏิบัติการคอมพิวเตอร์ของผู้ดูแลระบบให้มีประสิทธิภาพมากยิ่งขึ้น

1.2 ความมุ่งหมายและวัตถุประสงค์ของปัญหาพิเศษ

1.2.1 พัฒนาโปรแกรมที่มีความสามารถตรวจสอบการใช้งานเครื่องคอมพิวเตอร์ของนักศึกษาภายในห้องปฏิบัติการคอมพิวเตอร์เพื่ออำนวยความสะดวกให้แก่ผู้ดูแลระบบ โดยกำหนดรูปแบบและความสามารถที่โปรแกรมควรจะมีและมีส่วนการติดต่อโปรแกรม (User Interface) ที่เหมาะสมและมีมาตรฐานเพื่อให้ผู้ใช้เข้าใจได้อย่างถูกต้องและครบถ้วน

1.2.2 สามารถนำโปรแกรมที่ทางคณะผู้จัดทำได้จัดทำขึ้นไปใช้งานได้จริง

1.2.3 เพื่อให้นักศึกษาตระหนักถึงกฎระเบียบในการใช้งานห้องปฏิบัติการคอมพิวเตอร์

1.2.4 ศึกษาการเขียนโปรแกรมที่ทำงานติดต่อกันระหว่างเครื่องคอมพิวเตอร์ โดยสนับสนุนโปรโตคอล TCP/IP

1.2.5 ศึกษาการเขียนโปรแกรมที่เกี่ยวข้องกับการทำงานของแก่นวินโดวส์

1.2.6 สามารถประยุกต์เทคนิคและวิธีการจัดการโครงการเข้ากับการทำงานร่วมกันภายในกลุ่ม โดยให้ความสำคัญกับการทำงานร่วมกันเป็นทีมในการพัฒนาโปรแกรม ซึ่งในแต่ละขั้นตอนการพัฒนาส่วนต่าง ๆ ทีมงานจะมีการปรึกษากันอยู่ตลอดเวลาเพื่อให้เกิดการทำงานอย่างมีขั้นตอนและเป็นลำดับ สามารถแบ่งงานกันทำได้และกำหนดระยะเวลา และทรัพยากรที่จำเป็นต้องใช้ในการทำงานได้อย่างครบถ้วนและมีความเหมาะสม

1.2.7 ศึกษาการเขียนโปรแกรมด้วย Visual Studio C++ 6.0

1.3 ขอบเขตของการศึกษา

ขอบเขตของการศึกษาจะเป็นไปตามลักษณะความสามารถของโปรแกรมที่ได้กำหนดโดยโปรแกรมสามารถช่วยอำนวยความสะดวกแก่ผู้ดูแลระบบในการตรวจสอบการใช้งานเครื่องคอมพิวเตอร์ของนักศึกษา โดยที่ผู้ดูแลระบบไม่จำเป็นต้องลุกจากเครื่องควบคุมระบบของตนเลย (Remote Control Access)

โปรแกรมสามารถทำงานต่าง ๆ ตามลักษณะความสามารถที่ควรจะเป็น คือ

1.3.1 เครื่องควบคุมระบบที่ผู้ดูแลระบบใช้อยู่สามารถมองเห็นหน้าจอของเครื่องคอมพิวเตอร์ทุกเครื่องที่ต่ออยู่อย่างถูกต้องภายในระบบเครือข่ายของห้องปฏิบัติการคอมพิวเตอร์ เพื่อให้ผู้ดูแลระบบสามารถตรวจสอบการใช้งานของนักศึกษาได้ว่ากำลังใช้งานแอพพลิเคชันอะไร หรือกำลังเปิดเว็บไซต์อะไรอยู่ และเป็นไปตามกฎระเบียบในการใช้ห้องปฏิบัติการหรือไม่

1.3.2 สามารถส่งข้อความเตือนหรือโต้ตอบกันระหว่างเครื่องควบคุมระบบกับเครื่องคอมพิวเตอร์ลูกข่าย

1.3.3 สามารถตรวจจับและแสดงได้ว่าเครื่องคอมพิวเตอร์ลูกข่ายทุก ๆ เครื่องภายในห้องปฏิบัติการกำลังใช้งานแอพพลิเคชันใดอยู่บ้าง

1.3.4 ผู้ดูแลระบบสามารถสั่งปิดเครื่องคอมพิวเตอร์ลูกข่ายจากเครื่องควบคุมระบบได้ในกรณีที่เห็นว่าเหมาะสม

1.4 แผนการดำเนินงาน

ระยะเวลาในการดำเนินงานของทีมงาน ดังตารางที่ 1.1

แผนงาน / ระยะเวลา	มิ.ย.	ก.ค.	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.
กำหนดปัญหา วิเคราะห์ Requirement ที่ได้รวมรวบไว้ (Define Requirement)		1 ก.ค.- 2 ส.ค.								
กำหนดนิยามของซอฟท์แวร์ที่พัฒนาขึ้น (Specification)			3 ส.ค.- 9 .ส.ค							
กำหนดแผนงานและขั้นตอนในการดำเนินการต่าง ๆ และกำหนดหน้าที่ของสมาชิกในทีมงานตามความเหมาะสม(Planning)			10ส.ค.-13ส.ค.							
ออกแบบโปรแกรม(Design)			16ส.ค.-15ต.ค.							
เขียนโปรแกรม(Coding)					11ต.ค.-18ม.ค.					
รวมโปรแกรม (Integration)								19ม.ค.-13มี.ค.		
ติดตั้งและทดสอบโปรแกรม รวมถึงการปรับปรุงแก้ไขจนเสร็จสมบูรณ์ (Implementation)									15มี.ค.-30มี.ค.	

ตารางที่ 1.1 แสดงระยะเวลาในการดำเนินงาน

1.5 ขั้นตอนในการศึกษาและพัฒนาซอฟต์แวร์

1.5.1 ศึกษาระบบการใช้งานห้องปฏิบัติการคอมพิวเตอร์ของนักศึกษาและข้อมูลเกี่ยวกับกฎระเบียบต่าง ๆ ที่นักศึกษาต้องปฏิบัติเมื่อเข้ามาใช้บริการ เป็นการศึกษาระบบการทำงานและหน้าที่ความรับผิดชอบของผู้ดูแล โดยใช้เป็นแนวทางในการออกแบบและวิเคราะห์ความต้องการของผู้ดูแลระบบเพื่อให้สามารถนำโปรแกรมที่พัฒนาขึ้นไปใช้งานจริงได้

1.5.2 การทำงานร่วมกันของทีมงานจะใช้หลักการบริหารและจัดการโครงการในการกำหนดระยะเวลาของการทำงาน โดยมีการแบ่งขั้นตอนการทำงานออกเป็นส่วน ๆ อย่างชัดเจน เหมาะสมและสามารถแบ่งงานกันทำได้ภายในทีมงานแต่ละคน

1.5.3 เมื่อได้แนวทางและข้อกำหนดในการพัฒนาซอฟต์แวร์แล้ว จึงเริ่มพัฒนาจริงตามลำดับขั้นตอนดังต่อไปนี้

1) ศึกษาการเขียนโปรแกรมด้วย Visual C++ เวอร์ชัน 6.0 เพื่อนำมาใช้เป็นเครื่องมือในการพัฒนาโปรแกรม
2) การพัฒนาการติดต่อสื่อสารกันระหว่างเครื่องคอมพิวเตอร์โดยผ่านโปรโตรคอล TCP/IP
    1.1) ศึกษาและรวบรวมข้อมูลด้านการควบคุมการรับส่งข้อมูลผ่านระบบเครือข่าย
    1.2) ศึกษาและรวบรวมข้อมูลด้านการส่งแพ็กเก็ตของไฟล์ระหว่างเครื่องผ่านระบบเครือข่าย
    1.3) พัฒนาโปรแกรมในส่วนการติดต่อกันผ่านระบบเครือข่ายบนโปรโตคอล TCP/IP
    1.4) ทดลองติดต่อกันโดยเริ่มต้นด้วยการติดต่อด้วยแมสเสจเพียงอย่างเดียวก่อนและให้สามารถตรวจสอบวันที่ของเครื่องคอมพิวเตอร์ลูกข่าย และโต้ตอบแมสเสจกันได้ระหว่างเครื่องคอมพิวเตอร์ลูกข่ายกับเครื่องควบคุมระบบ
3) การพัฒนาการจับภาพหน้าจอคอมพิวเตอร์เครื่องคอมพิวเตอร์ลูกข่าย (Client Screen Capture)
4) การพัฒนาการสั่งให้เครื่องคอมพิวเตอร์ลูกข่ายหยุดระบบปฏิบัติการลง (Shutting Down Operating System)
5) การพัฒนาการตรวจจับแอพพลิเคชันหรือโปรแกรมที่รันอยู่บนคอมพิวเตอร์เครื่องคอมพิวเตอร์ลูกข่าย

1.5.4 พัฒนาโปรแกรมให้เครื่องควบคุมระบบสามารถรองรับเครื่องคอมพิวเตอร์ลูกข่ายได้มากกว่า 1 เครื่อง

1.5.5 ออกแบบหน้าจอของโปรแกรมฝั่งเครื่องคอมพิวเตอร์ลูกข่ายและเครื่องควบคุมระบบ

1.5.6 รวบรวมโปรแกรมที่พัฒนาทุก ๆ ส่วนเข้าด้วยกัน

1.5.7 ติดตั้งและทดสอบโปรแกรมในห้องปฏิบัติการจริง เพื่อดูข้อบกพร่องที่เกิดขึ้นเมื่อนำมาใช้งานในระบบเครือข่าย (network) เพื่อดูว่างานที่ได้เป็นไปตามที่ต้องการหรือไม่ และหากมีข้อตกลงหรือผลการตอบรับจากผู้ดูแลระบบว่าต้องการให้โปรแกรมมีความสามารถในด้านใดเพิ่มขึ้นอีก ทางทีมงานก็จะทำการศึกษาหาแนวทางที่จะปรับปรุงแก้ไขให้เสร็จสมบูรณ์ตามที่ผู้ดูแลระบบต้องการต่อไป

1.5.8 ปรับปรุงเปลี่ยนแปลงโปรแกรมให้ถูกต้อง ครบถ้วนและสมบูรณ์ตรงตามความต้องการของผู้ดูแลระบบมากที่สุด

1.6 ข้อตกลงเบื้องต้น

1.6.1 พิจารณาการควบคุมห้องปฏิบัติการคอมพิวเตอร์ โดยให้มีเครื่องควบคุมระบบเพียงหนึ่งเครื่องเท่านั้นและพิจารณาจำนวนเครื่องคอมพิวเตอร์ลูกข่ายตามที่ความสามารถของระบบเครือข่ายที่จะรองรับได้

1.6.2 ระบบสามารถทำงานได้อย่างสมบูรณ์ก็ต่อเมื่อโปรแกรม PCLC-Server และโปรแกรม PCLC-Client กำลังทำงานอยู่

1.6.3 โปรแกรมที่พัฒนาขึ้นจะทำหน้าที่เพียงตรวจจับและแสดงข้อมูลซึ่งจะเป็นแอพพลิเคชันหรือไฟล์รูปภาพเท่านั้น แต่จะไม่สามารถตัดสินใจที่จะปิดแอพพลิเคชันที่กำลังใช้งานอยู่ หรือปิด (shut down) เครื่องลูกข่ายนั้นได้ การทำงานตรงส่วนนี้จะทำโดยคำสั่งจากผู้ดูแลระบบเท่านั้น

1.6.4 ระบบจะต้องมีระบบรักษาความปลอดภัย ป้องกันมิให้นักศึกษาที่กำลังใช้งานอยู่ที่เครื่องคอมพิวเตอร์ลูกข่ายนั้นหาทางปิดโปรแกรมของระบบที่ฝังตัวทำงานอยู่ที่เครื่องคอมพิวเตอร์ลูกข่าย

1.6.5 ไม่พิจารณาความล้มเหลวของโปรแกรม อันเนื่องมาจากการทำงานผิดพลาดของระบบปฏิบัติการ เช่น ทรัพยากร (resource) ไม่เพียงพอ หรือความผิดพลาดที่เกิดจากระบบเครือข่าย

1.7 อุปกรณ์ที่ใช้ในการทำปัญหาพิเศษ

1.7.1 เครื่องคอมพิวเตอร์ที่ใช้ในการพัฒนาโปรแกรมมีคุณลักษณะดังนี้

- หน่วยประมวลผลกลางความเร็ว 350 MHz
- Harddisk 8.4 GB
- RAM 128 MB
- ระบบปฏิบัติการ windows 98 หรือ window 95 และ windows NT เซิร์ฟเวอร์ 4.0 Thai Enabled
- Visual Studio 6.0

1.7.2 ห้องปฏิบัติการคอมพิวเตอร์ที่มีเครื่องคอมพิวเตอร์ไม่ต่ำกว่า 2 เครื่อง และเชื่อมต่อกันด้วยระบบ LAN ผ่านโปรโตคอล TCP/IP โดยกำหนดให้เครื่องเซิร์ฟเวอร์ใช้ WindowsNT 4.0 เซิร์ฟเวอร์หรือดีกว่า ส่วนเครื่องไคลเอนต์ใช้ระบบปฏิบัติการ Windows 95 หรือ Windows 98 เพื่อใช้ในการตรวจสอบโปรแกรมในการใช้งานจริง ๆ

1.8 คำจำกัดความที่ใช้ในการศึกษา

1.8.1 ผู้ใช้โปรแกรม (User) คือ บุคคลที่เกี่ยวข้องกับซอฟต์แวร์ดังนี้

1) ผู้ดูแลระบบ คือผู้ใช้โปรแกรมตรวจสอบนักศึกษาที่เข้ามาใช้บริการห้องปฏิบัติการคอมพิวเตอร์ ว่าได้กระทำตามกฎระเบียบที่ได้กำหนดไว้หรือไม่ ตลอดจนพิจารณาว่าจะทำการปิดเครื่องคอมพิวเตอร์ของนักศึกษาผู้นั้นในกรณีที่เห็นว่าเหมาะสม

2) นักศึกษาที่เข้ามาใช้ห้องปฏิบัติการคอมพิวเตอร์ เป็นผู้ถูกตรวจสอบโดยโปรแกรมเพื่อความเป็นระเบียบเรียบร้อยของห้องปฏิบัติการคอมพิวเตอร์

1.8.2 เครื่องคอมพิวเตอร์

1) เครื่องควบคุม คือเครื่อง PC ที่ทำงานเป็นเครื่องควบคุมดูแลการใช้งานเครื่องคอมพิวเตอร์ของนักศึกษาในห้องปฏิบัติการคอมพิวเตอร์ใช้งานโดยผู้ดูแลห้องปฏิบัติการคอมพิวเตอร์

2) เครื่องคอมพิวเตอร์ลูกข่ายในระบบ มีหน้าที่ให้บริการแก่นักศึกษาที่เข้ามาใช้บริการในห้องปฏิบัติการคอมพิวเตอร์

1.8.3 โปรแกรมที่พัฒนา

1) PCLC-Server คือ โปรแกรมส่วนที่พัฒนาเพื่อให้ทำงานบนเครื่องควบคุม

2) PCLC-Client คือ โปรแกรมส่วนที่พัฒนาเพื่อทำงานบนเครื่องลูกข่าย

บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 โปรโตคอล TCP/IP

2.1.1 ความเป็นมาของโปรโตคอล TCP/IP

โปรโตคอล TCP/IP เป็นมาตรฐานของการรับส่งข้อมูลระหว่างคอมพิวเตอร์สองระบบที่มีจุดเริ่มต้นราว 30 ปีมาแล้ว เมื่อกระทรวงกลาโหมสหรัฐฯ หรือ Department Of Defense (DOD) ทำโครงการทดลองในปี ค.ศ 1969 เชื่อมโยงคอมพิวเตอร์ทางทหารของแต่ละหน่วย ซึ่งเป็นคอมพิวเตอร์ต่างชนิดกันให้สามารถติดต่อรับส่งข้อมูลกัน (File Transfer) และสามารถใช้บริการอื่นๆ เช่น Remote Login รวมถึงการรับส่งจดหมายอิเล็กทรอนิคส์ (E-mail) ด้วย จุดประสงค์ของโครงการนี้คือสร้างระบบเครือข่ายคอมพิวเตอร์ให้สามารถส่งข้อมูลกันได้ แม้ว่าสายส่งข้อมูลบางส่วนหรือคอมพิวเตอร์บางเครื่องในเครือข่ายจะถูกทำลายเสียหายไปก็ตาม ซึ่งเป็นคุณสมบัติที่สำคัญอย่างยิ่งเมื่อใช้งานยามเกิดสงคราม

ในขณะนั้นกองทัพบกเลือกใช้คอมพิวเตอร์และระบบเครือข่ายของ Digital Equipment Corporation (DEC) กองทัพเรือเลือกใช้คอมพิวเตอร์ของ Unisys ส่วนกองทัพอากาศเลือกใช้คอมพิวเตอร์ของ IBM เมื่อจะทำการรบ กระทรวงกลาโหมสหรัฐฯ ก็พบว่าคอมพิวเตอร์ของทั้ง 3 กองทัพสื่อสารข้ามระบบกันไม่ได้ จึงได้ให้ทุนในการทำโครงการเชื่อมต่อคอมพิวเตอร์ของทั้ง 3 กองทัพเข้าด้วยกันเป็นระบบเครือข่าย โดยมีคุณสมบัติพิเศษแตกต่างจากระบบเครือข่ายที่ใช้งานกันทั่ว ๆไปคือการรับส่งข้อมูลจะแบ่งข้อมูลออกเป็นส่วนย่อย ๆ เรียกว่า “แพ็กเก็ต” (packet) ข้อมูลแต่ละส่วนนี้จะถูกส่งไปให้คอมพิวเตอร์ผู้รับปลายทางผ่านสายส่งข้อมูล โดยแต่ละส่วนอาจใช้เส้นทางสำหรับส่งข้อมูลคนละทางก็ได้ คอมพิวเตอร์ปลายทางจะนำข้อมูลที่ได้รับมาต่อรวมกันตามลำดับจนครบ หากเส้นทางที่ส่งข้อมูลเสียหาย หรือเครื่องคอมพิวเตอร์บางส่วนในเครือข่ายเสียหาย ข้อมูลก็จะถูกส่งไปใหม่โดยใช้เส้นทางอื่นแทนโดยอัตโนมัติ โครงการนี้มีชื่อว่า Advanced Research Projects Agency Network หรือที่รู้จักกันดีในชื่อ ARPANET ซึ่งประสบความสำเร็จอย่างสูง จนใช้งานกันอย่างจริงจังในปี ค.ศ 1975

ซอฟต์แวร์ที่ใช้ควบคุมการรับส่งข้อมูลของ ARPANET ประกอบด้วยส่วนหลัก ๆ 2 ส่วนคือ

1. **TRANSMISSION CONTROL PROTOCOL** หรือ **TCP โปรโตคอล** มีหน้าที่ตรวจสอบการรับส่งข้อมูลระหว่างคอมพิวเตอร์ผู้รับและผู้ส่งให้ได้รับข้อมูลถูกต้องครบถ้วน และหากข้อมูลสูญหายก็จะแจ้งให้ต้นทางส่งข้อมูลเข้ามาใหม่

2. INTERNET PROTOCOL หรือ IP โปรโตคอล จะมีหน้าที่เลือกเส้นทางที่ใช้รับส่งข้อมูลผ่านระบบเครือข่าย และตรวจสอบ Address ของผู้รับโดยใช้ข้อมูลขนาด 4 ไบต์ หรือ 32 บิตเป็นตัวกำหนด Address ของผู้รับเรียกว่า IP Address

ต่อมาในปี ค.ศ 1983 TCP/IP ถูกกำหนดให้เป็นมาตรฐานการรับส่งข้อมูลของกระทรวงกลาโหมสหรัฐฯ เราจึงถือว่า TCP/IP มีต้นกำเนิดมาจากโครงการ ARPANET นั่นเอง และได้ถูกรวมเข้าเป็นส่วนหนึ่งของระบบปฏิบัติการ UNIX โดย Bolt Beranek และ Newman ซึ่งได้รับทุนสนับสนุนจากกระทรวงกลาโหมสหรัฐอีกเช่นเดียวกัน ทำให้ TCP/IP ก้าวเข้าสู่ระบบคอมพิวเตอร์ทางธุรกิจหลังจากใช้งานเฉพาะเครือข่ายของทางทหารมานานถึง 14 ปี

หลังจากที่สถาบันการศึกษาและมหาวิทยาลัยต่างๆในสหรัฐฯ เลือกใช้คอมพิวเตอร์ระบบปฏิบัติการ UNIX กันอย่างแพร่หลาย TCP/IP ก็ยิ่งมีบทบาทในการเชื่อมต่อคอมพิวเตอร์เข้าเป็นเครือข่ายมากขึ้นเรื่อย ๆ จนกระทั่งกลายเป็นอินเตอร์เน็ตในปัจจุบัน โดยมี ARPANET เป็นแกนกลาง และได้มีการกำหนดมาตรฐานที่ใช้ในการรับส่งข้อมูลเครือข่ายอื่น ๆ เพิ่มเติมขึ้นมาในภายหลังรวมทั้ง OSI Model ในเวลาต่อมา

2.1.2 โปรโตคอล TCP

โปรโตคอล TCP (Transmission Control Protocol) เป็นโปรโตคอลที่มีการรับส่งข้อมูลแบบ Stream Oriented Protocol หมายความว่าการรับส่งข้อมูลจะไม่คำนึงถึงปริมาณข้อมูลที่จะส่งไป แต่จะแบ่งข้อมูลเป็นส่วนย่อย ๆ ก่อน แล้วจึงจะส่งไปยังปลายทางอย่างต่อเนื่องเป็นลำดับข้อมูล ในกรณีที่ข้อมูลส่วนใดส่วนหนึ่งสูญหายไปก็จะส่งข้อมูลส่วนนั้นใหม่อีกครั้ง สำหรับปลายทางก็จะทำหน้าที่จัดเรียงส่วนของข้อมูลใหม่ให้ต่อเนื่องกันและประกอบกลับเป็นข้อมูลทั้งหมดได้ ซึ่งจะแยกข้อมูลที่ไม่ถูกต้องออก ดังนั้นแอพพลิเคชันหรือกระบวนการใดที่อาศัยการส่งผ่านข้อมูลด้วยโปรโตคอล TCP จะต้องใช้หน่วยความจำและขนาดของช่องสัญญาณ (Bandwidth) มากกว่า UDP

การติดต่อระหว่างกันจะต้องเป็นแบบ **Connection-Oriented** คือต้องมีการสร้างการติดต่อกันเป็น Session ทั้ง 2 ด้านเสียก่อน แล้วจึงจะรับส่งข้อมูลไปได้พร้อมกัน (Full Duplex) จากนั้นจึงเริ่มต้นติดต่อกัน และเมื่อต้องการจะเลิกการติดต่อก็จะต้องมีการยกเลิกการติดต่อ

ในระหว่างการรับส่งข้อมูลนี้โปรโตคอล TCP จะเพิ่มกระบวนการสอบทานข้อมูลเพื่อให้ข้อมูลมีความถูกต้องไม่ผิดพลาดไปจากเดิม โดยการส่งสัญญาณสอบทานข้อมูล (Acknowledgement) และส่งข้อมูลใหม่อีกครั้ง ถ้าปลายทางไม่ได้รับหรือเกิดความผิดพลาดขึ้น

2.1.3 โปรโตคอล IP

โปรโตคอล IP ทำหน้าที่ให้บริการส่งผ่านข้อมูลที่มาจาก Host-to-Host Layer เพื่อส่งข้ามไปยังเครือข่ายใด ๆได้อย่างถูกต้อง แม้ว่าจะมีเครือข่ายเชื่อมต่อกันอยู่ในอินเตอร์เน็ตเป็นล้าน ๆ เครือข่ายก็ตาม เนื่องจากโปรโตคอล IP มีข้อมูลตำแหน่ง IP ปลายทางที่จะส่งข้อมูลไปให้ โดยทำงานร่วมกับอุปกรณ์ Router เพื่อส่งข้อมูลข้ามเครือข่ายออกไปได้

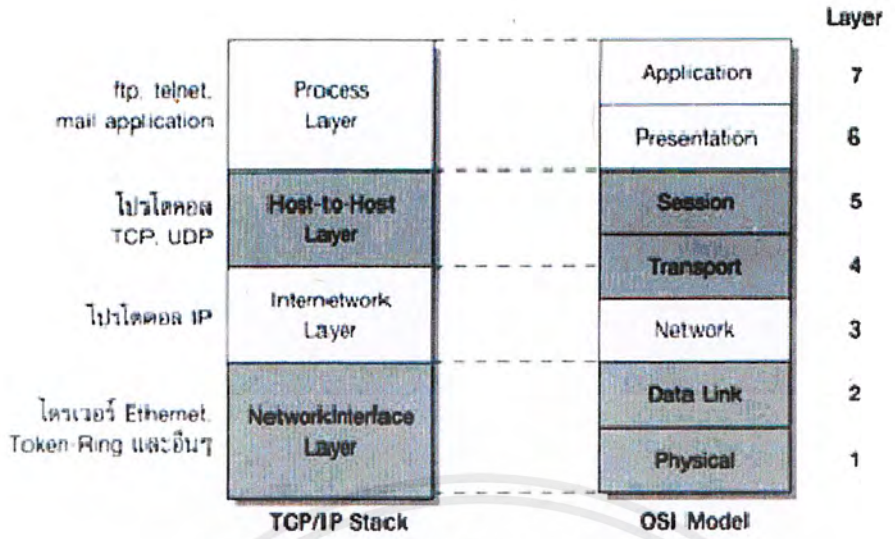
ตัวโปรโตคอล IP จะทำงานแบบ Packet Switching คือมีการส่งข้อมูลผ่านสวิตซ์ไปยังปลายทางตัววงจรผ่านหรือสวิตซ์อาจเป็น Gateway หรือ Router ในระบบเครือข่ายก็ได้ ซึ่งในข้อมูลของโปรโตคอล IP จะมีข้อมูลของหมายเลข IP ปลายทางที่จะส่งข้อมูลไปและเมื่อถึงเครือข่ายปลายทางแล้ว จะมีกลไกแปลงหมายเลข IP ให้เป็นหมายเลขฮาร์ดแวร์ประจำเครื่องที่ถูกต้องอีกทีหนึ่งด้วยโปรโตคอล ARP

2.1.4 โครงสร้างของโปรโตคอล TCP/IP

โปรโตคอล TCP/IP มีการจัดกลไกการทำงานเป็นชั้นหรือเลเยอร์เรียงต่อกันโดยในแต่ละเลเยอร์จะมีการทำงานเทียบได้กับ OSI model มาตรฐาน แต่บางเลเยอร์ของโปรโตคอล TCP/IP จะทำงานเทียบกับ OSI หลายเลเยอร์ของโปรโตคอล TCP/IP จะประกอบด้วย

- **Process Layer**
- **Host-to-Host Layer**
- **Internetwork Layer**
- **Network Interface Layer**

จากรูปที่ 2.1 เราจะเห็นว่าบางกลไกของโปรโตคอล TCP/IP เทียบได้กับมาตรฐาน OSI model สองชั้นหรือบางกลไกจะทำงานคาบเกี่ยวกันระหว่างบางชั้นของ OSI model ตัวอย่างเช่น กลไกการทำงานของโปรโตคอล TCP/IP ในส่วน Network Interface Layer เมื่อเทียบกับมาตรฐาน OSI model จะเทียบได้กับ Data Link Layer และ Physical Layer 2 ชั้นรวมกัน เป็นต้น ในแต่ละกลไกของโปรโตคอล TCP/IP จะมีโปรโตคอลอื่นๆ ในชุดของ TCP/IP ร่วมทำงานอยู่ด้วย

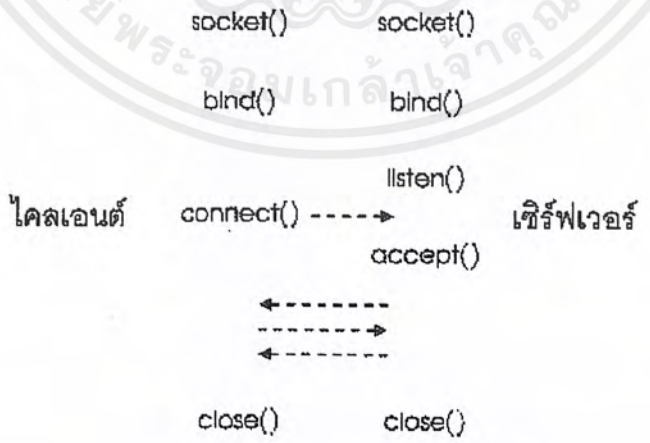


รูปที่ 2.1 แสดงการเปรียบเทียบเลเยอร์ของ TCP/ IP โปรโตคอลกับมาตรฐาน OSI model

2.2 แนวคิดการเขียนโปรแกรมเครือข่ายแบบไคลเอนต์ เซิร์ฟเวอร์

2.2.1 หลักการแบบไคลเอนต์ เซิร์ฟเวอร์

หลักการพื้นฐานมีอยู่ว่า การรับส่งข้อมูลผ่านเครือข่ายจะใช้สิ่งที่เรียกว่า **ซ็อกเก็ต (socket)** ซึ่งทั้งผู้รับและผู้ส่งจะต้องสร้างซ็อกเก็ตขึ้นมา ฝ่ายใดที่ต้องการข้อมูลจะต้องสั่งให้ซ็อกเก็ตของตนเชื่อมต่อกับซ็อกเก็ตของผู้รับ เมื่อซ็อกเก็ตของผู้รับยอมให้มีการเชื่อมต่อได้ ทั้งสองฝ่ายจึงจะเริ่มรับส่งข้อมูลระหว่างกันได้โดยการทำงานผ่านซ็อกเก็ตตลอด



รูปที่ 2.2 แสดงหลักการทำงานแบบไคลเอนต์ เซิร์ฟเวอร์

ถ้ามองในด้านโปรแกรมมิ่ง หลังจากที่โปรแกรมสร้างซ็อกเก็ตมาได้แล้วมี 2 กรณีที่เราสามารถเลือกสั่งให้ซ็อกเก็ตทำงานได้

1. กรณีแรกคือ ถ้าเราเขียนโปรแกรมให้เป็นเซิร์ฟเวอร์ ตอนนี้จะต้องสั่งให้ซ็อกเก็ตของเรา อยู่ในสถานะรอการร้องขอติดต่อจากซ็อกเก็ตอื่น ซ็อกเก็ตที่ร้องขอเข้ามาอาจจะอยู่ในเครื่องเดียวกัน เครื่องอื่น ๆ ในเครือข่ายเดียวกันหรือมาจากเครือข่ายอินเตอร์เน็ตก็ได้

2. กรณีที่สองคือ ถ้าเราเขียนโปรแกรมให้เป็นไคลเอนต์ เราจะต้องสั่งให้ซ็อกเก็ตที่สร้างขึ้น ทำการเชื่อมต่อไปยังซ็อกเก็ตเป้าหมายที่ต้องการ ซ็อกเก็ตเป้าหมายจะอยู่ในเครื่องเดียวกันหรือต่างเครื่องก็ได้ เมื่อซ็อกเก็ตในเซิร์ฟเวอร์ยอมรับการเชื่อมต่อแล้ว เราจึงจะสามารถสั่งซ็อกเก็ตของเราขอข้อมูลจากเซิร์ฟเวอร์ได้

เพื่อให้ง่ายต่อการทำความเข้าใจเรื่องของซ็อกเก็ต ไม่ว่าซ็อกเก็ตนั้นจะทำหน้าที่เป็นไคลเอ็นต์หรือเป็นเซิร์ฟเวอร์ก็ตามจะขอเปรียบเทียบการทำงานของซ็อกเก็ตกับโทรศัพท์ที่เราใช้อยู่ในปัจจุบัน ซึ่งมีขั้นตอนดังนี้

1. เตรียมเครื่องโทรศัพท์

ก่อนที่เราจะใช้โทรศัพท์ได้ เราต้องมีเครื่องโทรศัพท์ติดตั้งเตรียมพร้อมไว้ก่อน การเขียนโปรแกรมเครือข่ายก็เช่นเดียวกัน คือจะต้องเตรียมซ็อกเก็ตก่อน ซ็อกเก็ตก็เปรียบเสมือนตัวเครื่องโทรศัพท์ ขั้นตอนนี้จะเรียกใช้ฟังก์ชัน **socket()** เพื่อสร้างซ็อกเก็ตขึ้นมา

การติดต่อสื่อสารจะต้องใช้ซ็อกเก็ตทั้งฝ่ายผู้ส่งและผู้รับ ผู้ส่งต้องสร้างซ็อกเก็ตขึ้นมา และส่งข้อมูลไปยังผู้รับผ่านซ็อกเก็ต ส่วนผู้รับต้องสร้างซ็อกเก็ตขึ้นมารอรับข้อมูลด้วย ที่สำคัญคือในเครื่องหนึ่งๆสามารถมีซ็อกเก็ตได้หลายตัวและทำงานพร้อมกันได้ เพราะแต่ละซ็อกเก็ตจะมีพอร์ตทำงานแยกจากกันโดยอิสระ เหมือนกับบ้าน 1 หลังสามารถมีโทรศัพท์ได้หลายเลขหมาย การแยกความแตกต่างว่าซ็อกเก็ตใดทำงานอะไรหรือคอยข้อมูลจากซ็อกเก็ตไหนนั้น ซ็อกเก็ตแต่ละตัวจะมีหมายประจำตัวกำกับไว้ ซึ่งได้มาจาก IP แอดเดรสของเครื่องที่ซ็อกเก็ตนั้นรันอยู่ผสมกับหมายเลขพอร์ตที่ซ็อกเก็ตนั้นใช้งาน ซ็อกเก็ตจึงทำงานแยกจากกันอย่างอิสระ ไม่รบกวนซึ่งกันและกัน

พอร์ตที่ว่านี้เป็นพอร์ตทางซอฟแวร์ไม่ใช้พอร์ตที่เป็นฮาร์ดแวร์หลังตัวเครื่องคอมพิวเตอร์ พอร์ตเปรียบเสมือนท่อสำหรับส่งข้อมูลระหว่างไคลเอนต์กับเซิร์ฟเวอร์ พอร์ตจะใช้ตัวเลขขนาด 16 บิตกำกับไว้ ซึ่งหมายความว่าในแต่ละเครื่องจะมีหมายเลขพอร์ตให้ใช้งานถึง 65536 พอร์ต เริ่มจากพอร์ตหมายเลข 0 ถึง 65535 ในระดับการทำงานจริงๆ แล้ว พอร์ตเป็นเพียงตัวเลขที่จะถูก “ปะ“ ส่งไปกับข้อมูล เพื่อให้รู้ว่าข้อมูลนั้นจะทำงานกับพอร์ตหมายเลขอะไรในเซิร์ฟเวอร์ปลายทาง หมายเลขพอร์ตจึงเป็นตัวแยกข้อมูลไม่ให้ปนกัน

สมมติว่าเราเขียนโปรแกรมเพื่อทำงานเป็นเซิร์ฟเวอร์ รอการร้องขอข้อมูลจากไคลเอนต์ โปรแกรมเราจะต้องสร้างซ็อกเก็ตขึ้นก่อน ขั้นแรกนี้เราจะได้ซ็อกเก็ตหรือตัวเครื่องโทรศัพท์เตรียมพร้อมสำหรับการติดต่อแล้ว แต่ยังใช้รับสายไม่ได้เพราะยังไม่ได้เสียบสายเชื่อมกับใครเลย

2 เสียบสายโทรศัพท์

หลังจากที่เตรียมเครื่องโทรศัพท์แล้ว ต่อไปคือ เสียบสายจากตัวเครื่องโทรศัพท์เข้าคู่สายที่ผนังบ้านหรือเป็นการกำหนดแอดเดรสให้กับซ็อกเก็ต ขั้นตอนนี้จะใช้ฟังก์ชัน **bind()** ถ้าเป็นโทรศัพท์ก็เหมือนกับว่า มีคู่สายหลายคู่สายเข้ามาในบ้าน แต่ละคู่สายย่อมจะมีหมายเลขโทรศัพท์ประจำอยู่แล้ว เราเพียงแต่เลือกว่าจะเสียบตัวเครื่องโทรศัพท์ไปต่อกับคู่สายหมายเลขอะไร ถ้าเป็นซ็อกเก็ตก็หมายความว่าในตอนนี้ซ็อกเก็ตที่สร้างขึ้นมีแอดเดรสประจำตัวแล้ว

3. รอเรียกสาย

ถ้าเขียนโปรแกรมเพื่อทำงานเป็นเซิร์ฟเวอร์ ซ็อกเก็ตของเราจะต้องรอการร้องขอการเชื่อมต่อจากไคลเอนต์ การทำงานนี้ใช้คำสั่ง **listen()** ถ้าเป็นโทรศัพท์ก็หมายถึงกำลังรอเรียกสาย

4. เรียกสาย

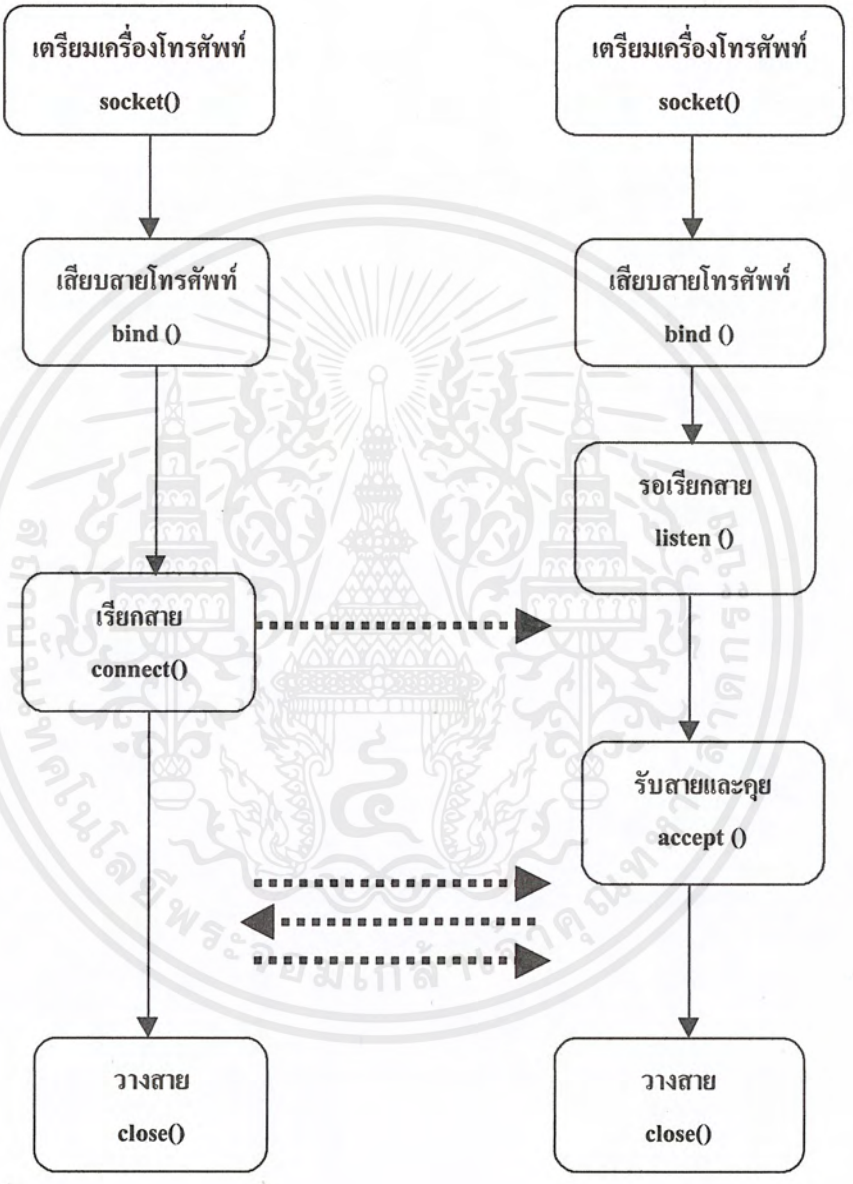
มาดูฝ่ายผู้โทรเรียกหรือไคลเอนต์กันบ้าง ฝ่ายนี้ก็ต้องเตรียมเครื่องโทรศัพท์แล้วเสียบสายจากตัวเครื่องโทรศัพท์เข้าสู่คู่สายเหมือนกัน นั้นคือการสร้างซ็อกเก็ตในเครื่องขึ้นมาก่อน แล้วกำหนดแอดเดรสให้ซ็อกเก็ต จากนั้นก็หาซ็อกเก็ตปลายทางที่จะเชื่อมต่อ แล้วจึงเชื่อมซ็อกเก็ตในเครื่องไปยังซ็อกเก็ตที่รออยู่ในเซิร์ฟฟเวอร์ปลายทาง ด้วยการใช้ฟังก์ชัน **connect()** สิ่งที่ต้องบอกแก่ฟังก์ชัน connect() คือ ชื่อของซ็อกเก็ตในเครื่องของเราและแอดเดรสของซ็อกเก็ตปลายทาง แอดเดรสของซ็อกเก็ตก็เปรียบเสมือนเบอร์โทรศัพท์ปลายทางที่เราจะโทรไปนั่นเอง ถ้าไม่ทราบหรือหาไม่ได้เราก็ไม่สามารถโทรศัพท์ไปยังเครื่องเป้าหมายปลายทางได้

5. รับสายและคุย

เมื่อเซิร์ฟเวอร์ในซ็อกเก็ตปลายทาง ได้รับสัญญาณการขอเชื่อมต่อและยอมรับการเชื่อมต่อด้วยคำสั่ง **accept()** แล้ว ทั้งไคลเอนต์และ เซิร์ฟเวอร์จึงเริ่มการติดต่อรับส่งข้อมูลระหว่างกันได้ หากเป็นโทรศัพท์ก็หมายถึงผู้รับได้ยินเสียงกริ่งโทรศัพท์ และยกหูเพื่อพูดคุยกับผู้เรียก

6. วางสาย

การรับส่งข้อมูลระหว่างไคลเอนต์กับเซิร์ฟเวอร์ จะกระทำผ่านซ็อกเก็ตได้เรื่อยๆ จนกว่าฝ่ายใดฝ่ายหนึ่งเป็นฝ่ายบอกตัดการเชื่อมต่อ ด้วยคำสั่ง **close()** เหมือนกับการพูดคุยกันทางโทรศัพท์ ซึ่งเราสามารถคุยได้นานจนกระทั่งฝ่ายใดวางหูโทรศัพท์ การเชื่อมต่อระหว่างทั้ง 2 ฝ่ายจึงจบลง



รูปที่ 2.3 แสดงขั้นตอนการทำงานของโทรศัพท์

2.3 หลักการในการตรวจจับแอพพลิเคชัน

2.3.1 โหลดไฟล์ Kernel32.dll ขึ้นมา ไฟล์ Kernel32.dll นี้จัดเป็นแก่นของวินโดวส์ซึ่งมีหน้าที่ในการจัดการทรัพยากร หน่วยความจำและโปรเซสต่าง ๆ ของวินโดวส์

2.3.2 ตรวจสอบแอพพลิเคชันที่กำลังใช้งานอยู่โดยใช้ฟังก์ชันCreateToolhelp32Snapshot

2.4 หลักการจับภาพ (Capture) หน้าจอของวินโดวส์

หลักการจับภาพหน้าจอของวินโดวส์มีหลักการดังต่อไปนี้

2.4.1 สร้าง device context ต้นทางและ device context ปลายทางขึ้นมาเพื่อเป็นสื่อกลางที่จะใช้ติดต่อกับอุปกรณ์แสดงผล ในที่นี้ก็คือจอมอนิเตอร์นั่นเอง เนื่องจากเราไม่สามารถที่จะเขียนโปรแกรมติดต่อกับอุปกรณ์ได้โดยตรง

2.4.2 เก็บรายละเอียดของภาพหน้าจอ

2.4.3 นำรายละเอียดของภาพหน้าจอไปใส่ไว้ใน device context ต้นทาง

2.4.4 ทำการคัดลอกและเคลื่อนย้ายรายละเอียดของภาพหน้าจอที่อยู่ใน device context ต้นทางไปยัง device context ปลายทาง

2.4.5 ทำการแปลงภาพ bitmap ประเภท DDB เป็น DIB

2.4.6 บันทึกไฟล์เป็นสกุล BMP

2.5 Bitmap

Bitmap เป็นวัตถุทางกราฟฟิกตัวหนึ่งที่มีความสามารถในการสร้าง , จัดการ , และเก็บภาพเป็นไฟล์ในดิสก์ โดย bitmap เป็น 1 ใน 7 ของวัตถุที่สามารถเลือกเข้าไปไว้ใน device context ได้ ซึ่งวัตถุอีก 6 ประเภทที่เหลือ ได้แก่ pen , brush , font , region , logical palette , และ drawing surface

จากมุมมองของผู้ใช้งานโดยทั่วไปนั้น bitmap เป็นเพียงภาพสี่เหลี่ยมที่ภายในประกอบไปด้วย pixels หลาย ๆ pixels มารวมตัวกันแล้วเกิดเป็นรูปภาพให้เห็นขึ้นมา แต่จากมุมมองของผู้พัฒนาแล้ว bitmap เป็นชุดของโครงสร้างที่ประกอบไปด้วยองค์ประกอบดังต่อไปนี้

- **Header**
- **Logical Palette**
- **Array of Bits**

Bitmap สามารถแบ่งออกได้เป็น 2 ประเภท คือ

2.5.1 Device-Independent Bitmaps (DIB) จะประกอบด้วยข้อมูลดังนี้คือ

- **Color Format**
- **Resolution**
- **Palette**
- **Array of Bits**
- **Data Compression Identifier**

คำอธิบายของประเภทข้อมูล DIB ดังตารางที่ 2.1

ประเภทข้อมูล	คำอธิบาย
1. color format	จะถูกระบุในรูปแบบของ color planes และ color bits per pixel
2. resolution	ความละเอียดของอุปกรณ์แสดงผล
3. palette	เป็นโครงสร้างที่ใช้เก็บข้อมูลเกี่ยวกับเรื่องของสี
4. array of bits	คือข้อมูลของ bitmap ที่มีการเก็บเป็น bits นั่นเอง
5. data compression identifier	แสดงถึงรูปแบบในการบีบอัดข้อมูลซึ่งใช้ในการลดขนาดของ array of bits

ตารางที่ 2.1 แสดงข้อมูล DIB

ซึ่งข้อมูลต่าง ๆ เหล่านี้จะถูกเก็บอยู่ในโครงสร้างที่เรียกว่า BITMAPINFO ซึ่งจะประกอบไปด้วยโครงสร้าง BITMAPINFOHEADER ซึ่งจะใช้ในการระบุถึงขนาดของ pixel rectangle , เทคโนโลยีของอุปกรณ์แสดงผล และรูปแบบของการลดขนาดของข้อมูล

นอกจากนี้ BITMAPINFO ยังประกอบไปด้วย RGBQUAD ซึ่งใช้ระบุสีที่ปรากฏอยู่ใน pixel rectangle อีกด้วย และเนื่องจากมีตารางสีเป็นของตัวเองด้วย ดังนั้นจึงสามารถทำงานเป็นอิสระกับอุปกรณ์แสดงผล หรือเรียกว่าเป็นอิสระต่อกันนั่นเอง

2.5.2 Device-Dependent Bitmaps (DDB)

เป็น bitmap ที่สนับสนุนเฉพาะกับแอพพลิเคชันที่เขียนขึ้นสำหรับวินโดวส์ในเวอร์ชันก่อนหน้าเวอร์ชัน 3.0 เท่านั้น ซึ่งสำหรับผู้ที่ต้องการพัฒนาแอพพลิเคชันใหม่ ๆ หรือต้องการนำแอพพลิเคชันที่ถูกเขียนขึ้นก่อนหน้าเวอร์ชั่น 3.0 ไปยังแพลตฟอร์มที่เป็นแบบ WIN 32 จะต้องใช้ bitmap แบบ DIBs bitmap ประเภทนี้ทำงานได้เฉพาะกับอุปกรณ์บางประเภทเท่านั้นไม่สามารถทำงานเป็นอิสระกับอุปกรณ์แสดงผลได้

2.6 Bitmap File Format

เป็น file format ภาพที่แสดงภาพได้เหมือนของจริงมากที่สุด เนื่องจากใช้ pixel เป็นองค์ประกอบพื้นฐานของภาพทำให้ใช้พื้นที่ในการเก็บมาก โดยขนาดของภาพจะขึ้นอยู่กับความละเอียดในการแสดงผลซึ่งเป็นจำนวนหน่วย pixel ต่อหนึ่งหน่วยพื้นที่ Window bitmap files จะถูกเก็บไว้ใน device-independent bitmap(DIB) format ที่อนุญาตให้ Windows แสดง bitmap บน display device ชนิดใดก็ได้ คำว่า "device independent" หมายถึง bitmap จะระบุ pixel color ในรูปแบบที่เป็นอิสระต่อวิธีการแสดงผลที่จะใช้แสดงสี

2.6.1 โครงสร้าง Bitmap-File

แต่ละ bitmap-file จะประกอบด้วย

1) bitmap-file header
2) bitmap-information header
3) color table
4) array of byte ที่ไว้แทน bitmap bits

ซึ่งมีโครงสร้างดังนี้

- BITMAPFILEHEADER bmfh;
- BITMAPINFOHEADER bmih;
- RGBQUAD *colormap;
- BYTE aBitmapBits[];

1) bitmap-file header

จะประกอบไปด้วยข้อมูลเกี่ยวกับชนิด และขนาดของ file รวมทั้งผังข้อมูลภายใน file ที่แสดงอยู่ใน field ต่าง ๆ ดังต่อไปนี้

```
typedef struct tagBITMAPFILEHEADER
{
        WORD        bfType;
        DWORD       bfSize;
        WORD        bfReserved1;
        WORD        bfReserved2;
        DWORD       bfOffBits;
} BITMAPFILEHEADER
```

ความหมาย

bfType	ระบุชนิดของ file โดยที่ค่าจะต้องเป็น "BM"(อักษร ASCII)
bfSize;	ระบุขนาดของ file (in bytes)
bfReserved1	ถูกสงวนไว้ต้อง set ค่าเป็น 0
bfReserved2	ถูกสงวนไว้ต้อง set ค่าเป็น 0
bfOffBits	ระบุ offset เป็น byte จาก BITMAPFILEHEADER structure สำหรับจุดเริ่มต้นของข้อมูล bitmap

2) bitmap-information header

จะระบุถึง มิติ (dimension) ชนิดของการบีบอัดข้อมูล (compression type) และข้อมูลสำหรับสี (color format) แสดงอยู่ใน field ต่าง ๆ ดังต่อไปนี้

```
typedef struct tagBITMAPINFOHEADER
{
        DWORD       biSize;
        LONG        biWidth;
        LONG        biHeight;
        WORD        biPlanes;
        WORD        biBitCount;
        DWORD       biCompression;
        DWORD       biSizeImage;
```

```
    LONG        biXPelsPerMeter;
    LONG        biYPelsPerMeter;
    DWORD       biClrUsed;
    DWORD       biClrImportant;
} BITMAPINFOHEADER
```

ความหมาย

biSize ระบุขนาด(byte)ของBITMAPINFOHEADER structure (40 byte)

biWidth ระบุความกว้างของ bitmap ในหน่วย pixel

biHeight ระบุความสูงของ bitmap ในหน่วย pixel

biPlanes ระบุจำนวนของระนาบ (planes) สำหรับ target device (มัก setให้=1)

biBitCount ระบุจำนวน bit per pixel(must be 1 ,4 , 8 or 24)โดย

Value	Meaning
1	Bitmap เป็น monochrome (สีเดียวหรือขาว-ดำ) และ color table จะประกอบด้วย 2 entries แต่ละ bit ใน array ของ bitmap จะแทนแต่ละ pixel ถ้า bit ถูก clear แล้ว pixel จะแสดงด้วยสีของ first entry ใน color table ถ้า bit ถูก set แล้ว pixel จะแสดงสีของ second entry ใน color table
4	Bitmap มีสีมากที่สุด แต่ละ pixel ใน bitmap จะแทนด้วย 4 bit เพื่อใช้ค้นหาสีใน color table ดังตัวอย่าง ถ้า first byte ใน bitmap เป็น 0X1F แล้ว byte นี้จะใช้แทน 2 pixel โดยที่ first pixel จะประกอบด้วยสีใน second table entry และ second pixel จะประกอบด้วยสีใน sixteenth table entry
8	Bitmap มีสีมากที่สุด 256 สี แต่ละ pixel ใน bitmap จะแทนด้วย 1-byte เพื่อใช้ค้นหาใน color table ดังตัวอย่างถ้า first byte ใน bitmap เป็น 0X1F แล้ว first pixel จะแสดงสีของ thirty-second table entry

Value	Meaning
24	Bitmap มีสีมากที่สุด 2^24 (224 สีถึงมากกว่า 16 ล้านสี) สมาชิก bmiColors จะเป็น NULL และแต่ละ 3-byte ที่เรียงอยู่ติดกันใน array ของ bitmap จะใช้แทนความเข้มของสี แดง เขียว และน้ำเงิน (ตามลำดับของ 3-byte) สำหรับแสดงสี 1 pixel

ตารางที่ 2.2 แสดงความหมายของ biBitCount

biCompression ระบุชนิดของการบีบขนาดข้อมูล ซึ่งมีค่าได้ค่าใดค่าหนึ่งดังต่อไปนี้

value	Meaning
BI_RGB	มีค่าเท่ากับ 0 ระบุว่า bitmap ไม่ถูกบีบขนาดข้อมูล(ใช้ข้อมูลทางด้านเกี่ยวกับ RGB)
BI_RLE8	มีค่าเท่ากับ 1 ระบุ run-length encoded format for bitmaps with 8 bits per pixel
BI_RLE4	มีค่าเท่ากับ 2 ระบุ run-length encoded format for bitmaps with 4 bits per pixel

ตารางที่ 2.3 แสดงความหมายของ biCompression

biSizeImage	ระบุขนาดของภาพมีหน่วยเป็น byte และจะถูกตั้งค่าเป็น 0 ถ้า bitmap อยู่ใน BI_RGB format
biXPelsPerMeter	ระบุความละเอียดตามแนวขวางในหน่วย pixel/meter ของ target device สำหรับแสดง bitmap(ใช้ค่านี้เพื่อตั้งค่าที่ดีที่สุดของ current device)
biYPelsPerMeter	ระบุความละเอียดตามแนวดิ่งในหน่วยpixel/meter ของ target device สำหรับแสดง bitmap
biClrUsed	ระบุจำนวนของสีที่ใช้ (จำนวนของ color indexes) ใน color table ที่ถูกใช้เพื่อแสดง bitmap ถ้าค่าเป็น 0 แล้ว bitmap จะได้จำนวนสีมากที่สุดที่ตรงกับค่าของ biBitCount

biClrImportant	ระบุจำนวนของ color indexes ที่ถูกพิจารณาว่ามีความสำคัญในการแสดงภาพ bitmap ถ้าค่านี้เป็น 0 แล้ว ทุกสีถือว่าสำคัญหมด ซึ่งตัวควบคุมนี้มีประโยชน์มากเมื่ออุปกรณ์แสดงผลไม่สามารถแสดงผลจำนวนสีทั้งหมดได้ แต่สามารถแสดงเฉพาะสีที่มีความสำคัญได้ ค่านี้ไม่มีความหมายสำหรับ 24 bitmap files (ข้อมูลเกี่ยวกับสีใน color table จะใช้เนื้อที่ 4 bytes โดยที่สีน้ำเงิน เขียว และแดง จะใช้สีละ 1 bytes ส่วน byte ที่เหลือจะมีค่าเป็น 0 (reserved)

3) color table

ถูกแทนด้วย array of RGBQUAD sturcture ซึ่งประกอบไปด้วยสมาชิกจำนวนมากเท่ากับสีที่มีได้ใน bitmap และ color table จะไม่ถูกนำมาใช้แสดง bitmap ด้วย 24 color bits เพราะการที่มีเนื้อหาไม่เพียงพอของ aColors array และแต่ละ pixel จะถูกแสดงค่าของ red-green-blue(RGB) เพื่อแสดง 24 color bit สีต่างๆในตารางควรจะอยู่ในลำดับของความสำคัญ ซึ่งสิ่งนี้จะช่วยให้ display driver สามารถแสดงสีของ bitmap บน device ที่ไม่สามรถแสดงสีได้มากทั้งหมดเท่าที่มีอยู่จริงในรูป bitmap ได้ color table มีรูปแบบเป็น array ของ RGBQUAD ซึ่ง RGBQUARD ซึ่งมีโครงสร้างต่อไปนี้

```
typedef struct  tagRGBQUAD
{
        BYTE  rgbBlue;          //ระบุถึงความเข้มของสีน้ำเงินในสี
        BYTE  rgbGreen;         //ระบุถึงความเข้มของสีเขียวในสี
        BYTE  rgbRed;           //ระบุถึงความเข้มของสีแดงในสี
        BYTE  rgbReserved;      //ไม่ใช้ จะต้อง set ค่าเป็น 0
}
```

color table ไม่จำเป็นต้องถูกทำให้สมบูรณ์ เช่น BMP files ซึ่งจะใช้ข้อมูลเกี่ยวกับสีแบบ 8 bits ไม่จำเป็นต้องระบุสีในทั้งหมด 256 สี ถ้า image ไม่ต้องการสีทั้งหมด

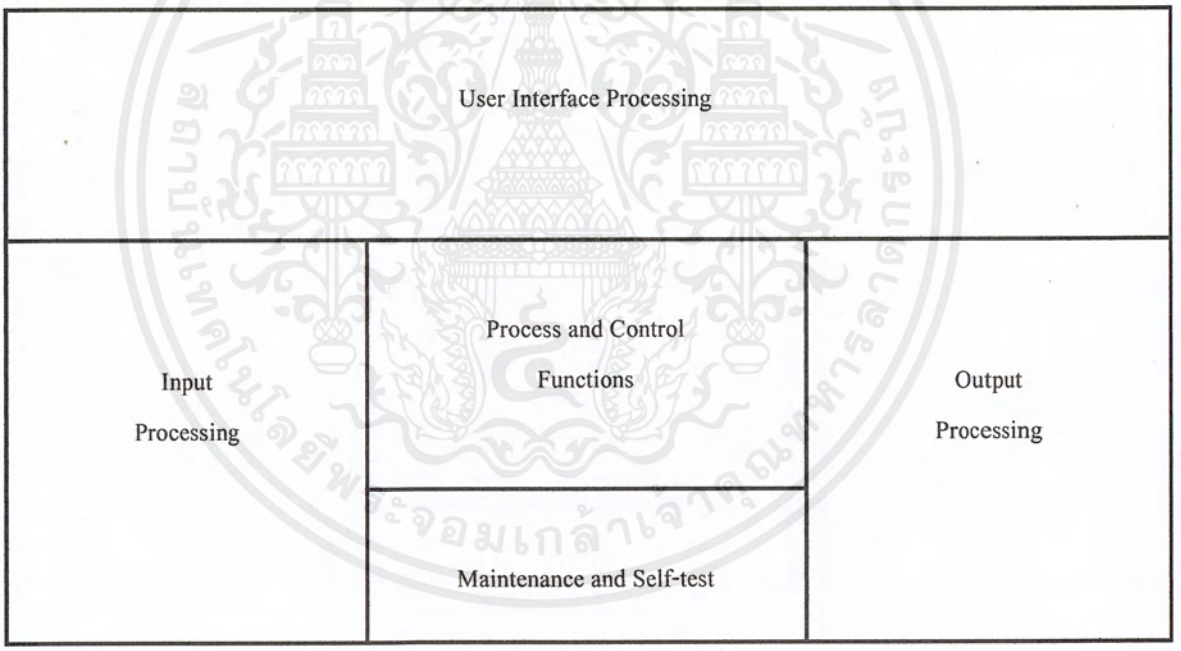
บทที่3

ภาพรวมของระบบ

3.1 สถาปัตยกรรมระบบทางกายภาพ (Physical System Architecture)

จากระบบของปัญหาพิเศษที่ทีมงานได้ทำการศึกษาสามารถนำมาเขียนให้อยู่ในรูปของสถาปัตยกรรมระบบทางกายภาพ ซึ่งแบ่งออกเป็นส่วนหลัก ๆ ได้ 5 ส่วน ดังนี้

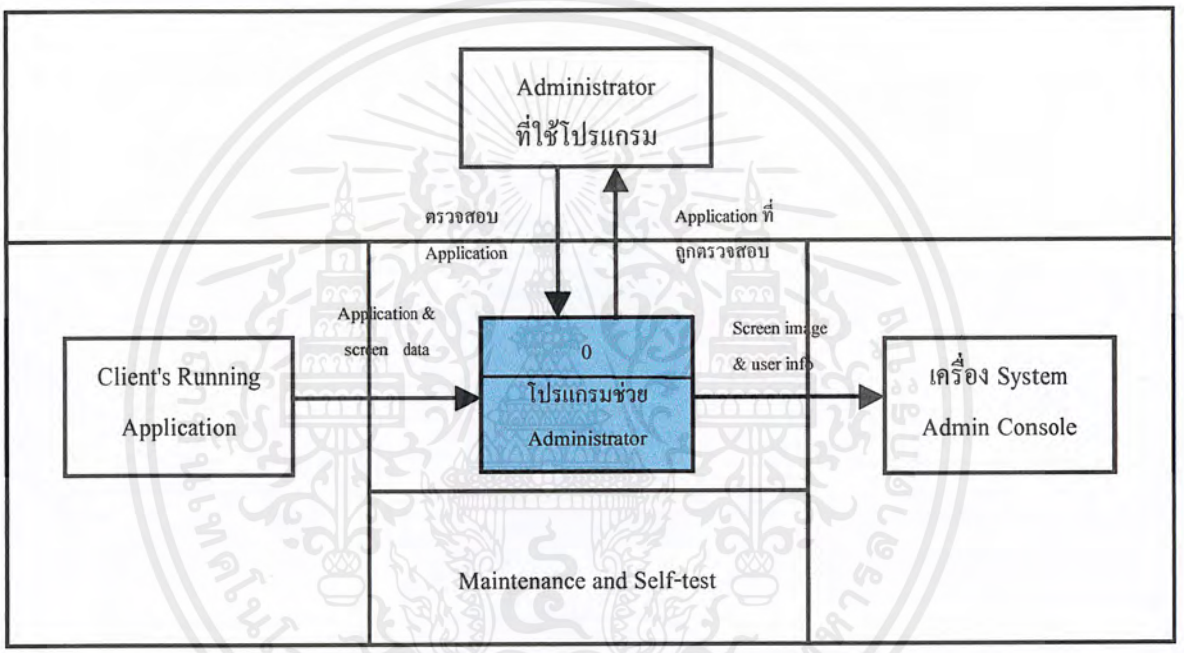
- **User Interface Processing**
- **Input Processing**
- **Process and Control Functions**
- **Output Processing**
- **Maintenance and Self-test**



รูปที่ 3.1 แสดง Architecture Template

เมื่อทำการพิจารณาระบบเพื่อจัดตามรูปแบบสถาปัตยกรรมทางกายภาพจะได้ว่า

- ส่วน User Interface Processing คือ ผู้ที่ทำการติดต่อกับโปรแกรมที่เราพัฒนาขึ้นนั้นคือ ผู้ควบคุมระบบ (Administrator)
- ส่วนที่จะเป็น Input Processing ให้แก่ระบบคือ เครื่องลูกข่าย
- ส่วนทำหน้าที่เป็น Output Processing คือ เครื่องควบคุม
- ส่วน Process and Control Functions คือ ตัวโปรแกรมที่เราพัฒนาขึ้น
- ส่วน Maintenance and Self-test เป็นส่วนที่ในระบบเราไม่ได้จัดทำขึ้น

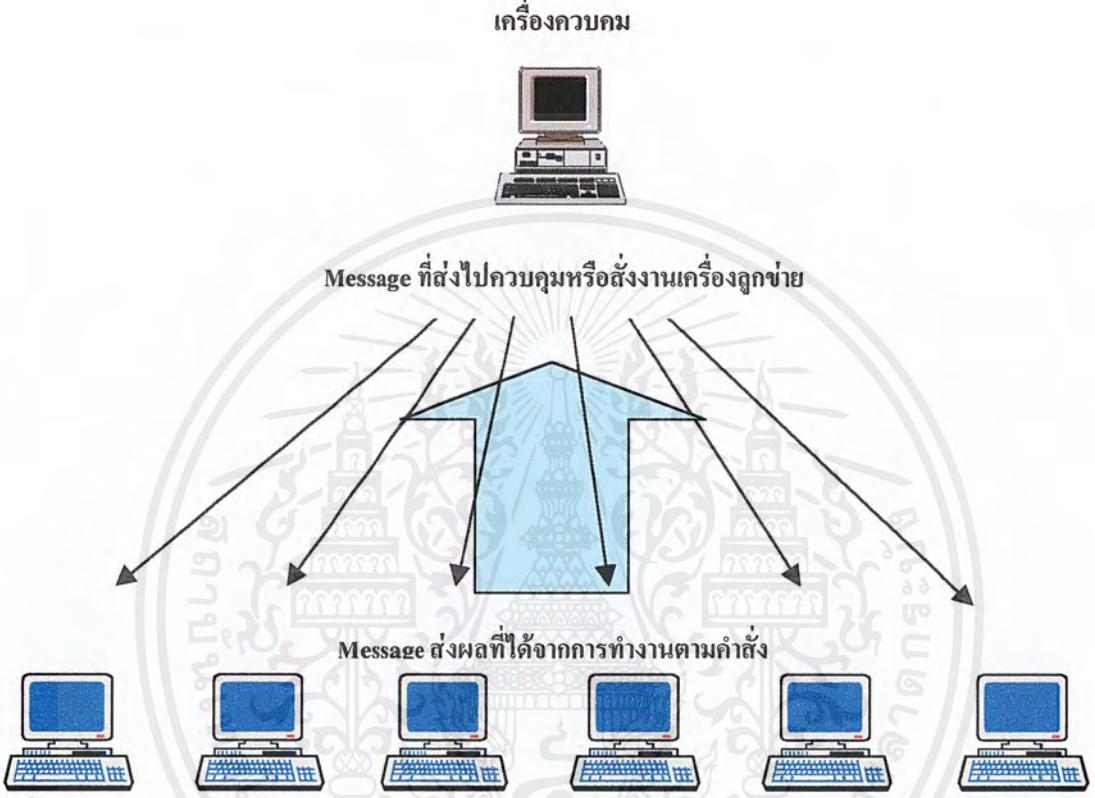


รูปที่ 3.2 แสดง System Architecture : Physical ของระบบ

3.2 ภาพจำลองของระบบ

โปรแกรมในระบบที่ได้พัฒนาขึ้นจะต้องแบ่งออกเป็น 2 ส่วนด้วยกันคือ ส่วนที่ทำงานบนเครื่องควบคุม และโปรแกรมส่วนที่ทำงานบนเครื่องคอมพิวเตอร์ลูกข่าย โดยโปรแกรมที่ทำงานบนเครื่องควบคุมจะทำหน้าที่คอยควบคุมและส่งคำสั่งไปยังเครื่องคอมพิวเตอร์ลูกข่ายที่อยู่ในระบบในรูปของแมสเสจ โดยมีหมายเลข IP ของเครื่องลูกข่ายในการอ้างอิง เมื่อคอมพิวเตอร์ลูกข่ายที่ได้อ้างอิงถึงได้รับแมสเสจที่เครื่องควบคุมส่งมา ก็จะทำการประมวลผลแมสเสจที่ได้รับว่าคืออะไร แล้วทำตามแมสเสจที่ได้รับ เช่น เครื่องควบคุมทำการส่งแมสเสจไปยังเครื่องที่ IP 192.168.0.4 ให้ทำการจับภาพหน้าจอ เมื่อเครื่องลูกข่ายได้รับแมสเสจจากเครื่องควบคุมก็จะทำการจับภาพหน้าจอแล้วทำ

การบันทึกเป็นไฟล์สกุล BMP จากนั้นทำการส่งไฟล์กลับไปยังเครื่องควบคุม การติดต่อระหว่างเครื่องควบคุมกับเครื่องลูกข่ายสามารถแสดงได้ดังรูปที่ 3.3



รูปที่ 3.3 แสดงภาพจำลองของระบบ

3.3 ฟังก์ชันการทำงานหลักของเครื่องควบคุม

หน้าที่การทำงานหลัก ๆ ของเครื่องควบคุม จะเป็นการส่งแมสเสจไปยังเครื่องลูกข่าย โดยในส่วนรายละเอียดของฟังก์ชันการทำงานหลักของเครื่องควบคุมจะกล่าวถึงอีกครั้งในบทที่ 5 เรื่องการออกแบบเครื่องควบคุม สำหรับการส่งแมสเสจจากเครื่องควบคุมสามารถจำแนกได้ดังนี้

3.3.1 ส่งคำสั่งการทำงานในรูปแบบของแมสเสจไปยังเครื่องลูกข่าย โดยคำสั่งที่ส่งไปจะอยู่ในรูปแบบใดรูปแบบหนึ่งต่อไปนี้

1) สั่งให้เครื่องลูกข่ายทำการจับภาพหน้าจอ

2) สั่งให้เครื่องลูกข่ายทำการจับแอพพลิเคชันที่กำลังทำงานอยู่ในขณะนั้นของเครื่องลูกข่าย

3) ส่งข้อความไปแสดงยังหน้าจอของเครื่องลูกข่าย

3.3.2 ส่งคำสั่งควบคุมในรูปแบบของแมสเสจไปยังเครื่องลูกข่าย โดยคำสั่งควบคุมจะอยู่ในรูปแบบเดียวคือ สั่งปิดระบบปฏิบัติการของเครื่องลูกข่าย

3.4 ฟังก์ชันการทำงานหลักของเครื่องลูกข่าย

การทำงานหลักของเครื่องลูกข่ายโดยส่วนใหญ่จะเป็นไปในลักษณะของการประมวลผลตามแมสเสจที่ได้รับจากเครื่องควบคุม ซึ่งสามารถเป็นได้อย่างใดอย่างหนึ่งดังต่อไปนี้ โดยรายละเอียดจะกล่าวถึงอีกครั้งในบทที่ 6 การออกแบบเครื่องลูกข่าย

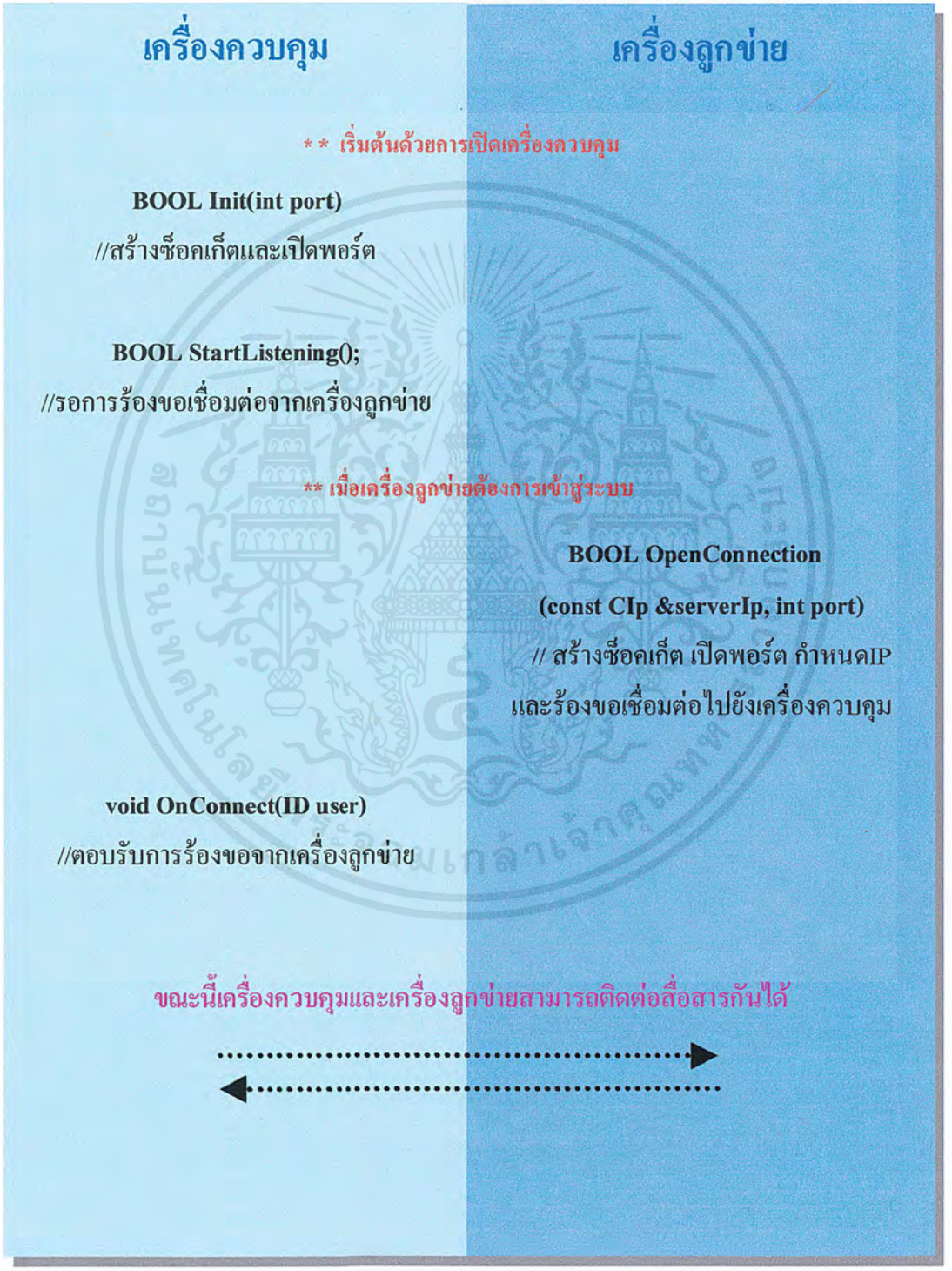
3.4.1 จับภาพหน้าจอแล้วบันทึกเป็นไฟล์สกุล BMP แล้วส่งไฟล์ไปยังเครื่องควบคุม

3.4.2 จับแอพพลิเคชันที่กำลังใช้งานอยู่ขณะนั้นแล้วบันทึกลงไฟล์เป็นสกุล TXT จากนั้นทำการเปิดไฟล์ขึ้นมาอ่านเพื่อทำการส่งข้อมูลมาแสดงยังเครื่องควบคุมทีละบรรทัด

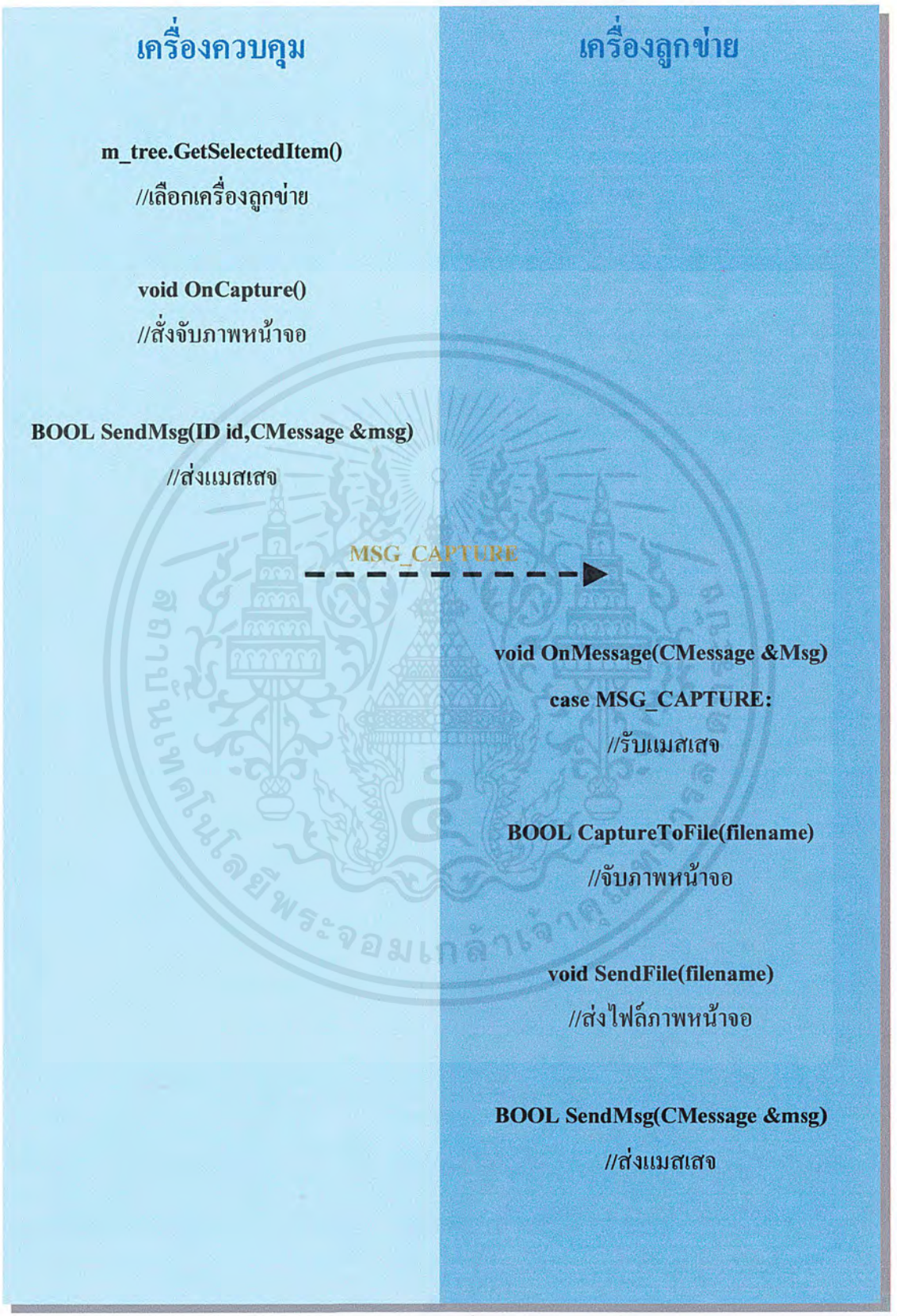
3.4.3 ทำการปิดระบบปฏิบัติการของตนเองลง

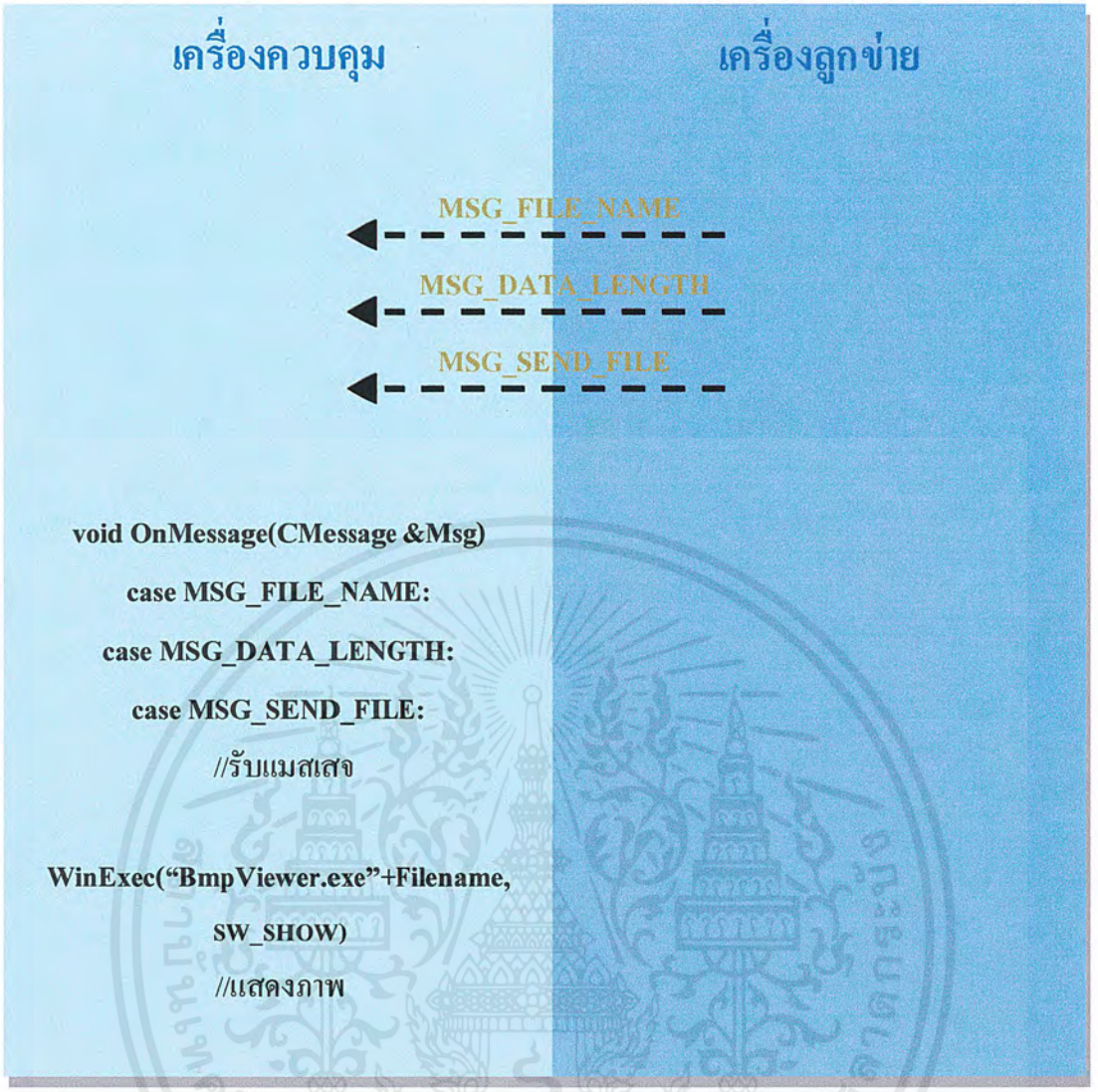
บทที่ 4
ความสัมพันธ์ระหว่างเครื่องควบคุมและเครื่องลูกข่าย

4.1 การติดต่อกันระหว่างเครื่องควบคุมกับเครื่องลูกข่าย

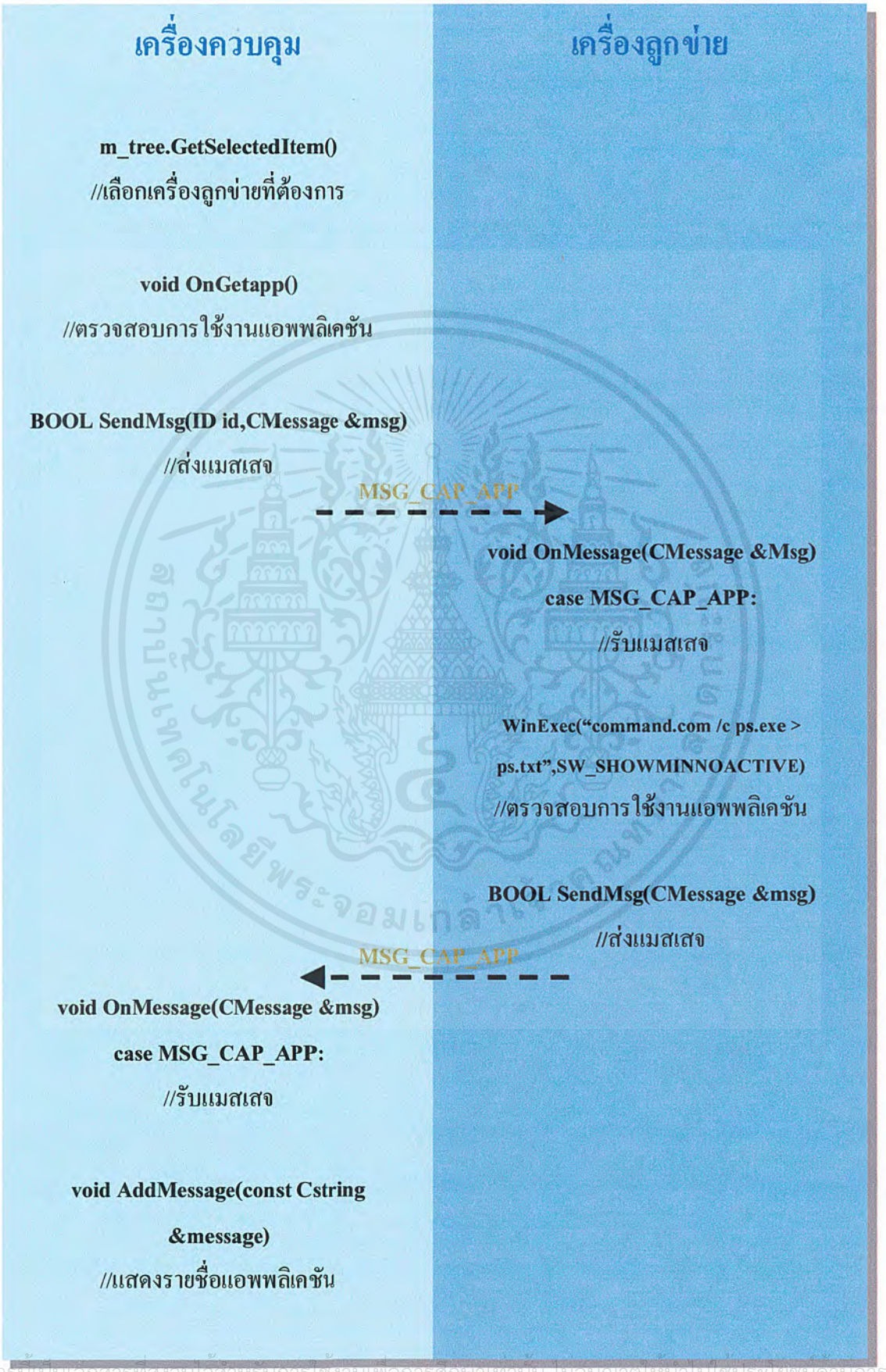


4.2 กรณีที่เครื่องควบคุมต้องการจับภาพหน้าจอของเครื่องลูกข่าย





4.3 กรณีที่เครื่องควบคุมต้องการตรวจสอบการใช้งานแอพพลิเคชันของเครื่องลูกข่าย



4.4 กรณีที่เครื่องควบคุมต้องการปิดการทำงานของเครื่องลูกข่าย

4.5 กรณีที่เครื่องควบคุมต้องการส่งข้อความติดต่อกับเครื่องลูกข่าย

บทที่ 5
การออกแบบโปรแกรม PCLC-Server

แนวคิดในการออกแบบโปรแกรม PCLC-Server แบ่งเป็น 2 ส่วน สำคัญคือ

- ส่วนการติดต่อกับผู้ใช้โปรแกรม
- ส่วนฟังก์ชันการทำงานของโปรแกรม

5.1 ส่วนการติดต่อกับผู้ใช้โปรแกรม

ส่วนการติดต่อกับผู้ใช้หรือผู้ดูแลระบบของโปรแกรม PCLC-Server ดังรูปที่ 5.1 ประกอบด้วยส่วนสำคัญ 4 ส่วนคือ

1) ส่วน Server Status เป็นส่วนแสดงสถานะของเซิร์ฟเวอร์และระบบในปัจจุบันว่ากำลังทำอะไรอยู่ ตลอดจนแสดงการร้องขอเพื่อที่จะติดต่อเข้าสู่ระบบเครือข่าย (connect) หรือตัดการติดต่อออกจากระบบเครือข่าย (disconnect) ของเครื่องคอมพิวเตอร์ลูกข่ายใด ๆ ภายในระบบโดยมีลำดับที่ในการอ้างถึงตามลำดับการเข้ามาในระบบ

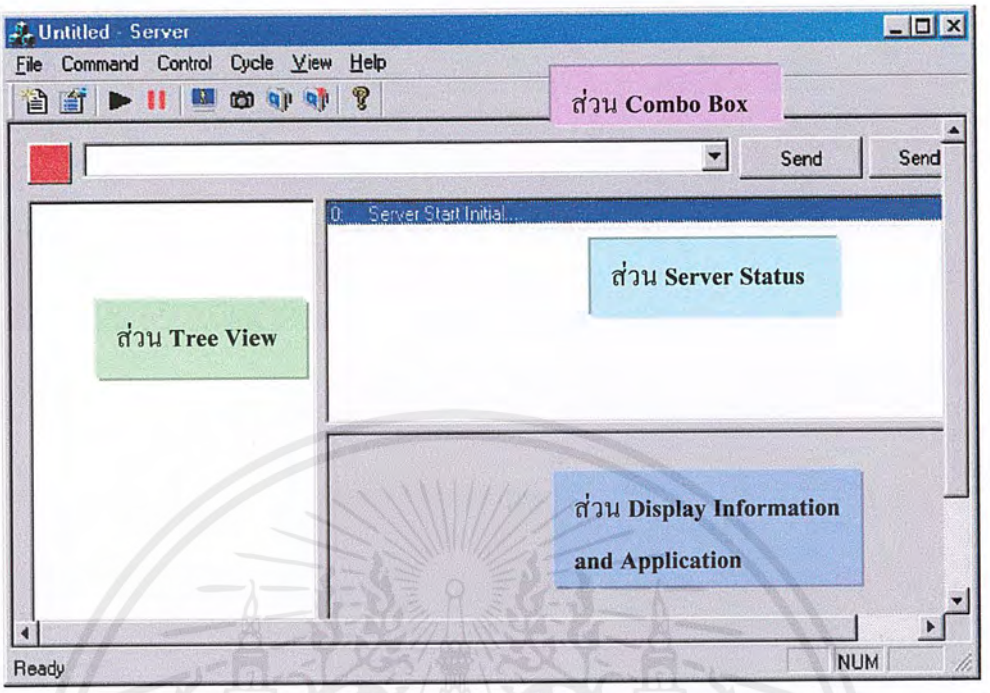
2) ส่วน Display Information and Application เป็นส่วนแสดงข้อมูลเกี่ยวกับสภาวะแวดล้อมของเครื่องคอมพิวเตอร์ลูกข่ายในระบบที่กำลัง Active ในขณะนั้นซึ่งอาจเป็นเครื่องควบคุมหรือเครื่องลูกข่ายใด ๆ ก็ได้ โดยสภาวะแวดล้อมในที่นี้ได้แก่

- หมายเลข IP
- หมายเลข Port ที่ใช้ในการติดต่อ
- ระบบปฏิบัติการ
- เวลาที่เข้าสู่ระบบ ฯลฯ

นอกจากนี้ในส่วนนี้ยังถูกสงวนไว้ใช้ในการแสดงรายชื่อของแอพพลิเคชันที่กำลังใช้งานอยู่บนเครื่องลูกข่ายใด ๆ ในขณะนั้น

3) ส่วน TreeView แสดงเครื่องคอมพิวเตอร์ในระบบ (node ต่างๆ) ในรูปของแผนภูมิต้นไม้ โดยมีการอ้างถึงเครื่องลูกข่ายใด ๆ ตาม IP ของเครื่องลูกข่ายนั้น ๆ

4) Combo Box ใช้ในการป้อนข้อความที่ต้องการส่งไปยังเครื่องลูกข่ายใดๆ ในระบบ นอกจากนี้ยังสามารถส่งข้อความไปให้กับเครื่องลูกข่ายทุกเครื่องในระบบพร้อมกัน ได้อีกด้วย



รูปที่ 5.1 แสดงส่วนการติดต่อกับผู้ใช้ของโปรแกรม PCLC-Server

5.2 ส่วนฟังก์ชันการทำงานของโปรแกรม PCLC-Server

การทำงานในส่วนของเครื่องควบคุมโดยโปรแกรม PCLC-Server จะทำงานเป็นขั้น ๆ ดังต่อไปนี้

5.2.1 การเตรียมพร้อมโปรแกรม PCLC-Server สำหรับการติดต่อ

5.2.2 ยอมรับการเชื่อมต่อจากเครื่องลูกข่ายเข้าสู่ระบบ

5.2.3 ควบคุมการทำงานของเครื่องลูกข่าย

5.2.4 ตัดการเชื่อมต่อกับเครื่องลูกข่าย

5.2.1 เตรียมพร้อมโปรแกรม PCLC-Server สำหรับการติดต่อ

การเริ่มต้นการทำงานสิ่งแรกที่จะต้องทำคือ **การดักฟังสัญญาณ** จากเครื่องลูกข่ายที่จะทำการติดต่อ (Connect) เข้าสู่ระบบ หมายถึงต้องเริ่มการทำงานทันทีเมื่อเริ่มต้นโปรแกรม เราจึงใส่โค้ดไว้ในส่วนการเริ่มต้น (Startup) ของโปรแกรมดังรูปที่ 5.2 นี้

```
void CServerView::OnInitialUpdate()
{
        CFormView::OnInitialUpdate();
        GetParentFrame()->RecalcLayout();
        ResizeParentToFit();

        CServerView::InitialServer(); //initial server listening
}
```

รูปที่ 5.2 แสดงฟังก์ชัน **OnInitialUpdate();**

โดยจะมีฟังก์ชันที่ทำหน้าที่ตรงส่วนนี้คือ **InitialServer();** ซึ่งจะแบ่งการทำงานเป็น 2 ขั้นตอน

1) การกำหนดพอร์ตที่จะดักฟังสัญญาณโดยฟังก์ชัน **Init(Port)**

2) เมื่อเรากำหนดพอร์ตที่เราทำการดักฟังแล้วก็จะเริ่มทำการเปิดพอร์ต เพื่อทำการดักฟัง โดยตัวฟังก์ชันจะทำการกำหนด IP ของเครื่องเป็น IP ของเครื่องควบคุมให้โดยอัตโนมัติ ส่วนนี้จะเป็นการทำงานของฟังก์ชัน **StartListening();** ซึ่งโค้ดโปรแกรมแสดงดังรูปที่ 5.3

```
BOOL CServerObj::StartListening()
{
        return m_pListeningSocket->Listen();
}
```

รูปที่ 5.3 แสดงฟังก์ชัน **StartListening();**

โดยโค้ดส่วนอื่นๆจะเป็นส่วนของการทำงานในการแสดงรายละเอียดของเครื่องควบคุมบนส่วนแสดงผล

5.2.2 ยอมรับการเชื่อมต่อจากเครื่องลูกข่ายเข้าสู่ระบบ

เมื่อผ่านขั้นตอนแรกแล้วเครื่องควบคุมก็พร้อมที่จะให้บริการในด้านต่าง ๆ ที่จะเกิดขึ้น และพร้อมที่จะรับเครื่องลูกข่ายที่จะติดต่อเข้ามา เพื่อที่จะสามารถดำเนินการต่าง ๆ ได้ โดยเครื่องควบคุมจะทำ **การดักรอ** เราสามารถจำแนกการดักรอออกเป็น 2 แบบคือ

1) การดักรอเครื่องลูกข่ายที่จะติดต่อเข้ามา เครื่องควบคุมจะสามารถควบคุมหรือติดต่อกับเครื่องลูกข่ายได้ก็ต่อเมื่อได้ทำการติดต่อเข้ามาในระบบเรียบร้อยแล้วเท่านั้น โดยเมื่อเครื่องลูกข่ายติดต่อเข้าสู่ระบบ เครื่องควบคุมก็จะทำการแปลง IP ของเครื่องลูกข่ายเป็นเลขลำดับในการติดต่อเข้าสู่ระบบ (ID User) เพื่อให้ง่ายในการเรียกใช้งานและเก็บไว้ไม่ให้ซ้ำกัน โดยฟังก์ชันที่ทำหน้าที่ดักรอนี้คือ **OnConnect (ID user);** ซึ่งแสดงโค้ดดังรูปที่ 5.4

```
void CServerView::OnConnect(ID user)
{
        sndPlaySound("message.wav",SND_ASYNC);
        CString temp;
        temp.Format("[User#%d] Connected...", user);
        SetTopic(temp);
        insert.hParent = htreeitem;
        insert.item.iImage=1;                                   //set client image
        insert.item.iSelectedImage=1;
        char client[20];
        itoa(user,client,10);
        insert.item.pszText=client;
        m_tree.InsertItem(&insert);
        m_tree.Expand(htreeitem,TVE_EXPAND);    //show all client
        LastUserID = user;
        InterruptCycle = true;
}
```

รูปที่ 5.4 แสดงฟังก์ชัน **OnConnect (ID user);**

เมื่อเครื่องลูกข่ายทำการติดต่อเข้าสู่ระบบ ฟังก์ชัน **OnConnect(ID User);** จะถูกเรียกใช้และส่งค่า user ที่ติดต่อเข้ามาคืนมาให้ โดยเราจะนำค่า user นี้ไปแปลงเพื่อให้ในการเก็บรายละเอียดต่าง ๆ ของเครื่องลูกข่ายแต่ละเครื่องในตัวแปร UserList ซึ่งเป็นอาร์เรย์ (Array) เก็บรายละเอียดของเครื่องลูกข่ายทั้งหมดที่อยู่ในระบบ ส่วนโค้ดส่วนอื่น ๆ นั้นเกี่ยวกับการแสดงผลให้ปรากฏบน TreeView

2) การดักฟังแมสเสจที่เครื่องลูกข่ายส่งมา โดยส่วนมากแล้วจะเกิดจากเครื่องควบคุมส่งแมสเสจร้องขอไปแล้ว เครื่องลูกข่ายก็จะตอบรับตามที่เครื่องควบคุมร้องขอไปกลับมา ก็คือการทำงานในส่วนนี้นั่นเอง เพราะฉะนั้นส่วนนี้จึงมีการทำงานในหลาย ๆ ด้าน โดยเมื่อมีการส่งแมสเสจมาจากเครื่องคอมพิวเตอร์ลูกข่าย ฟังก์ชัน **OnMessage();** จะถูกเรียกใช้งานพร้อมกับมีการคืนค่าพารามิเตอร์มา 2 ค่า คือ

- **ID User** บอกให้รู้ว่าแมสเสจนี้ถูกส่งมาจากเครื่องคอมพิวเตอร์ลูกข่ายคนไหน
- **Message** หรือข้อมูลที่ถูกส่งมาจากเครื่องคอมพิวเตอร์ลูกข่ายนั้น ๆ โดยมีโค้ดดังรูปที่ 5.5

```
void CServerView::OnMessage(ID user, CMessage &msg)
{
	switch (msg.GetIdentifier())
	{
		case MSG_SEND_TEXT:
		{
			CString text;
			CString result;
			msg.GetElementAt(0, text);
			result.Format("[User#%d] %s", user, text);
			ClearMessage();
			AddMessage(result);
		}
		break;
		case MSG_USER_JOIN:
		{
			char UserID[5];
			itoa(user,UserID,10);
			HTREEITEM htreeitem,hchild;
			htreeitem = m_tree.GetRootItem();
			hchild = m_tree.GetChildItem(htreeitem);
			while (m_tree.GetItemText(hchild) != UserID )
				hchild = m_tree.GetNextItem(hchild,TVGN_NEXT);
			CString tmp;
			msg.GetElementAt(0,tmp);
			m_tree.SetItemText(hchild,tmp);
			AddUser(user,tmp);
			ClearMessage();
			for (int i=0; i < msg.GetNbElements(); i++)
			{
				CString temp;
				msg.GetElementAt(i,temp);
				AddMessage(temp);
			}
			SaveHistory();
			InterruptCycle = false;
		}
		break;
		case MSG_USER_QUIT:
		{
			DisconnectUser(user);
			CString buffer;
			buffer.Format("[User#%d] is disconnected...", user);
			SetTopic(buffer);
```

```
                        SaveHistory();
                        CString UserID;
                        UserID = GetUserIDstr(user);
                        HTREEITEM htreeitem,hchild;
                        htreeitem = m_tree.GetRootItem();
                        hchild = m_tree.GetChildItem(htreeitem);
                        while (m_tree.GetItemText(hchild) != UserID )
                                hchild = m_tree.GetNextItem(hchild,TVGN_NEXT);
                        m_tree.DeleteItem(hchild);
                        DeleteUser(user);
                }
                break;
                case MSG_SEND_FILE: // get data area
                {
                        byte* data = new byte[dataLength];
                        msg.GetElementAt(0, data,dataLength);
                        CFile destFile(FileName, CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);
                        destFile.Write(data, dataLength); // Write it
                        destFile.Close();
                        delete data;
                        bar.SetText("Receiving Data");
                        for (int i = 0; i < 3000; i++) {        // perform operation
                                bar.StepIt();
                                PeekAndPump();
                        }
                        bar.Clear();
                }
                break;
                case MSG_DATA_LENGTH: //get length of data
                {
                        CString text;
                        msg.GetElementAt(0, text);
                        dataLength = atoi(text);
                }
                break;
                case MSG_FILE_NAME:
                {
                        msg.GetElementAt(0, FileName);
                        FileName.MakeLower();
                        bar.SetText("Capture Screen ");
                        if (FileName.Find(".bmp") != -1)
                        {
                                bar.SetText("Capture Screen ");
        for (int i = 0; i < 1500; i++) {
        // perform operation
```

```
}
                                        CString CommandLine("BmpViewer.exe " + FileName);
                                        WinExec(CommandLine,SW_SHOW);
                                }
                        }
                        break;
                        case MSG_CAP_APP:
                        {
                                int index;
                                index = msg.GetNbElements();
                                char c[10];
                                itoa(index,c,10);
                                for(int i=0;i<index;i++)
                                {
                                        CString app;
                                        msg.GetElementAt(i,app);
                                        AddMessage(app);
                                }
                                SaveHistory();
                        }
                        break;
                }
}
```

รูปที่ 5.5 แสดงฟังก์ชัน OnMessage();

จากโค้ดจะเห็นได้ว่าไม่ว่าจะเป็นการทำงานใด ๆ ก็ตามที่เครื่องคอมพิวเตอร์ลูกข่ายติดต่อเข้ามาจะอยู่ในส่วนของฟังก์ชัน **OnMessage();** นี้ทั้งหมด ซึ่งหมายถึงว่าจะมีแมสเสจหลาย ๆ ประเภทที่ถูกส่งเข้ามาและเราจะจำแนกแต่ละประเภทเป็นการทำงานแบบหนึ่ง หรือการทำงานอย่างหนึ่งอาจต้องประกอบด้วยการส่งแมสเสจมากกว่า 1 ครั้งก็เป็นได้ เช่น การส่งไฟล์ เป็นต้น โดยสามารถอธิบายการทำงานแยกเป็นกรณีอ้างอิงจากโค้ดที่แสดงข้างต้น ดังนี้

1. MSG_SEND_TEXT

แมสเสจประเภทนี้จะถูกเรียกใช้เมื่อเครื่องคอมพิวเตอร์ลูกข่ายทำการส่งแมสเสจที่เป็นข้อความ โดยมีจุดประสงค์หลักเพื่อให้เครื่องคอมพิวเตอร์ลูกข่ายสามารถส่งข้อความใด ๆ ก็ได้ที่ต้องการมาให้ผู้ดูแลระบบรับรู้ความต้องการของตน โดยข้อความที่ถูกส่งมาจะถูกแสดงในส่วนแสดงผลของเครื่องควบคุมดังโค้ด

```
case MSG_SEND_TEXT:
{
        CString text;
        CString result;
        msg.GetElementAt(0, text);
        result.Format("[User#%d] %s", user, text);
        ClearMessage();
        AddMessage(result);
}
break;
```

รูปที่ 5.6 แสดง **MSG_SEND_TEXT**

2. MSG_USER_JOIN

เมื่อเครื่องคอมพิวเตอร์ลูกข่ายสามารถติดต่อเข้ามายังเครื่องควบคุมได้เรียบร้อยแล้วนั้น เครื่องลูกข่ายนั้น ๆ จะส่งแมสเสจประเภทหนึ่งตามมาด้วย โดยมีจุดประสงค์หลัก ๆ เพื่อบอกรายละเอียดต่าง ๆ ของเครื่องลูกข่ายให้มาแสดงผลบนเครื่องควบคุม เช่น IP เวลา เป็นต้น ดังนั้นแมสเสจประเภทนี้จะถูกส่งมาทุกครั้ง เมื่อเครื่องลูกข่ายสามารถติดต่อเข้ามายังเครื่องควบคุมได้ ดังโค้ด

```
case MSG_USER_JOIN:
{
        char UserID[5];
        itoa(user,UserID,10);
        HTREEITEM htreeitem,hchild;
        htreeitem = m_tree.GetRootItem();
        hchild = m_tree.GetChildItem(htreeitem);
        while (m_tree.GetItemText(hchild) != UserID )
                hchild = m_tree.GetNextItem(hchild,TVGN_NEXT);
        CString tmp;
        msg.GetElementAt(0,tmp);
        m_tree.SetItemText(hchild,tmp);
        AddUser(user,tmp);
        ClearMessage();
        for (int i=0; i < msg.GetNbElements(); i++)
        {
                CString temp;
                msg.GetElementAt(i,temp);
                AddMessage(temp);
        }
        SaveHistory();
        InterruptCycle = false;
}
break;
```

รูปที่ 5.7 แสดง **MSG_USER_JOIN**

เมื่อเครื่องควบคุมรับแมสเสจนี้แล้ว สิ่งหลัก ๆ ที่ต้องทำคือเก็บรายละเอียดต่าง ๆ ลงใน UserList แล้วนำไปแสดงผลยังส่วนต่าง ๆ ให้ถูกต้อง เช่น การแสดง IP ของเครื่องลูกข่ายที่เข้ามาในระบบในส่วน TreeView

3. MSG_USER_QUIT

แมสเสจประเภทนี้จะถูกส่งมาจากเครื่องลูกข่าย เมื่อจะทำการตัดการติดต่อเครื่องลูกข่ายออกจากระบบ โดยจะถูกส่งมา 2 กรณี คือ เมื่อปิดระบบปฏิบัติการของเครื่องลูกข่ายลง และเมื่อสั่งปิดโปรแกรม PCLC-Client โดยเมื่อได้รับแมสเสจนี้ เครื่องควบคุมก็จะทำการเรียกฟังก์ชัน DisconnectUser เพื่อทำการลบเครื่องลูกข่ายออกจาก UserList และรายละเอียดของเครื่องลูกข่ายที่แสดงใน TreeView หรือส่วนแสดงรายละเอียดอื่น ๆ ออกไปดังโค้ด

```
case MSG_USER_QUIT:
{
DisconnectUser(user);
CString buffer;
buffer.Format("[User#%d] is disconnected...", user);
SetTopic(buffer);
SaveHistory();
CString UserID;
UserID = GetUserIDstr(user);
HTREEITEM htreeitem,hchild;
htreeitem = m_tree.GetRootItem();
hchild = m_tree.GetChildItem(htreeitem);
while (m_tree.GetItemText(hchild) != UserID )
hchild = m_tree.GetNextItem(hchild,TVGN_NEXT);
m_tree.DeleteItem(hchild);
DeleteUser(user);
}
break;
```

รูปที่ 5.8 แสดง **MSG_USER_QUIT**

4. MSG_FILE_NAME

แมสเสจประเภทนี้จะทำการรับชื่อไฟล์ที่ถูกส่งเข้ามาจากเครื่องลูกข่าย โดยจะรับมาเป็นไฟล์สกุล ZIP ซึ่งจะนำมาแตกในภายหลัง

```
case MSG_SEND_FILE: // get data area
{
        byte* data = new byte[dataLength];
        msg.GetElementAt(0, data,dataLength);
        CFile destFile(FileName, CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);
        destFile.Write(data, dataLength); // Write it
        destFile.Close();
        delete data;
        bar.SetText("Receiving Data");
        for (int i = 0; i < 3000; i++) {          // perform operation
                bar.StepIt();
                PeekAndPump();
        }
        bar.Clear();
}
break;
```

รูปที่ 5.9 แสดง MSG_FILE_NAME

5. MSG_DATA_LENGTH

เมื่อเครื่องควบคุมรับชื่อไฟล์ที่ถูกส่งมาจากเครื่องลูกข่ายแล้ว เราจำเป็นที่จะต้องรู้ต่อไปว่าขนาดของไฟล์นั้นเป็นเท่าไร แมสเสจประเภทนี้จึงทำหน้าที่ในการเก็บความยาวของไฟล์ที่จะถูกส่งมาเพื่อจะถูกนำไปใช้ต่อไป โดยมีโค้ดต่อไปนี้

```
case MSG_DATA_LENGTH: //get length of data
{
        CString text;
        msg.GetElementAt(0, text);
        dataLength = atoi(text);
}
break;
```

รูปที่ 5.10 แสดง MSG_DATA_LENGTH

6. MSG_SEND_FILE

เมื่อเราได้ข้อมูลที่จำเป็นต่อการจัดเก็บไฟล์คือชื่อไฟล์และขนาดของไฟล์มาเรียบร้อยแล้ว ขั้นตอนสุดท้ายเครื่องลูกข่ายก็จะทำการส่งข้อมูลในไฟล์มาให้โดยใช้แมสเสจ MSG_SEND_FILE นี้

ทำให้ในตอนนี้เราได้ข้อมูลที่จำเป็นครบหมดแล้ว และเราก็จะทำการบันทึกข้อมูลหล่านี้ลงไปในฮาร์ดดิสก์ โดยมีโค้ดดังต่อไปนี้

```
case MSG_SEND_FILE: // get data area
{
        byte* data = new byte[dataLength];
        msg.GetElementAt(0, data,dataLength);
        CFile destFile(FileName, CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);
        destFile.Write(data, dataLength); // Write it
        destFile.Close();
        delete data;
        bar.SetText("Receiving Data");
        for (int i = 0; i < 3000; i++) {        // perform operation
                bar.StepIt();
                PeekAndPump();
        }
        bar.Clear();
}
break;
```

รูปที่ 5.11 แสดง MSG_SEND_FILE

ซึ่งเป็นการทำงานของการบันทึกไฟล์ลงฮาร์ดดิสก์และแสดงผลความก้าวหน้าของการบันทึกให้เห็นโดยใช้ Progress Bar

7. MSG_CAP_APP

แมสเสจประเภทนี้จะถูกส่งมาโดยเครื่องลูกข่าย เมื่อเครื่องควบคุมสั่งให้มีการตรวจจับแอพพลิเคชัน จะมีลักษณะของแมสเสจพิเศษออกไป คือจะส่งออกมาเป็นอาร์เรย์ (Array) ของรายชื่อแอพพลิเคชันต่างๆ ที่กำลังใช้งานอยู่บนเครื่องลูกข่าย ทำให้เวลาที่ต้องการแสดงผลแอพพลิเคชันออกมาทางส่วนแสดงผลจำเป็นต้องมีอัลกอริทึมพิเศษเพื่อทำการแสดงผลโดยต้องทำการรับแอพพลิเคชันมาทีละ 1 ไปเรื่อย ๆ จนหมดดังโค้ดต่อไปนี้

```
case MSG_CAP_APP:
{
        int index;
        index = msg.GetNbElements();
        char c[10];
        itoa(index,c,10);
        for(int i=0;i<index;i++)
        {
                CString app;
                msg.GetElementAt(i,app);
                AddMessage(app);
        }
        SaveHistory();
}
break;
```

รูปที่ 5.12 แสดง **MSG_CAP_APP**

5.2.3 ควบคุมการทำงานของเครื่องลูกข่าย

จะแบ่งการทำงานออกเป็น 2 ลักษณะที่ทำงานไปพร้อม ๆ กัน คือ **การทำงานแบบอัตโนมัติ (Auto Operation)** และ **การทำงานตามปกติ (Manual Operation)** โดยจะแบ่งอธิบายกันดังนี้

1) การทำงานแบบอัตโนมัติ (Auto Operation) คือลักษณะการทำงานที่เครื่องควบคุมจะคอยตรวจสอบเครื่องลูกข่ายทุกเครื่องที่อยู่ในระบบ ภายในระยะเวลาที่กำหนด จากตัวแปร Cycle Time ซึ่งจะเป็นตัวกำหนดระยะเวลาหน่วงในแต่ละเครื่อง Auto Operation จะทำงานแค่การตรวจจับแอพพลิเคชัน และจะทำงานเอง โดยอัตโนมัติทุกครั้งที่เริ่มต้นโปรแกรมโดยประกาศเป็นฟังก์ชันแบบเทรด (thread) ซึ่งมีลักษณะดังรูปที่ 5.13

```
CServerView::CServerView()
        : CFormView(CServerView::IDD)
{
        m_command = _T("");
        m_event_text = _T("");
        insert.hParent=NULL;
        insert.hInsertAfter=TVI_LAST;
        CycleTime = 10; //set default
        InterruptCycle = true;
        LastUserID = 0;
        unsigned long idThread;
        m_hListenerThread = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)ServerAutoOperation,
(void *)this,0,&idThread);
        if(m_hListenerThread)
                AddMessage("Succeeded Listering...");
        else
                AddMessage("Failed...");
}
CServerView::~CServerView()
{
        if(m_hListenerThread)::TerminateThread(m_hListenerThread,0); //killing thread
        Reset();
}
```

รูปที่ 5.13 แสดงฟังก์ชันของการทำงานแบบเทรด

การทำงานแบบเทรดหมายถึงการสั่งให้ฟังก์ชันที่ถูกเรียกใช้นั้นๆทำงานเป็นอิสระโดยอัตโนมัติ โดยจะมีตัวแปรติดตามประเภท HANDLE คอยควบคุมอยู่เพื่อให้โปรแกรมหลัก (Main Program) สามารถติดต่อกับเทรดได้ โดยฟังก์ชันที่เรียกใช้คือ **ServerAutoOperation();** มีลักษณะดังรูปที่ 5.14

```
long WINAPI ServerAutoOperation(CServerView* pView)
{
	UINT count,CurrentUser;
	count = 1;
	CurrentUser = 1;
	CString timestring("");
	unsigned long NextTime;
	NextTime = 0;
	while (1) //infinite loop
	{
		if (!pView->InterruptCycle) //when not stop auto operation
		{
			while (pView->GetUserIDstr(CurrentUser) == "")
			{
				CurrentUser++;
				if (CurrentUser > pView->LastUserID)
					CurrentUser = 1;

			}
			CString u;
			u.Format("%d",CurrentUser);
			time_t tnow;
			time(&tnow);
			timestring.Format("%ld",tnow);
			if (NextTime == 0)
			{
				pView->CaptureApp(CurrentUser);
				NextTime = tnow + pView->CycleTime;
			}
			else if (tnow >= NextTime)
			{
				pView->CaptureApp(CurrentUser);
				NextTime += pView->CycleTime;
			}
			CurrentUser++;
		}
	}
}
```

รูปที่ 5.14 แสดงฟังก์ชัน **ServerAutoOperation();**

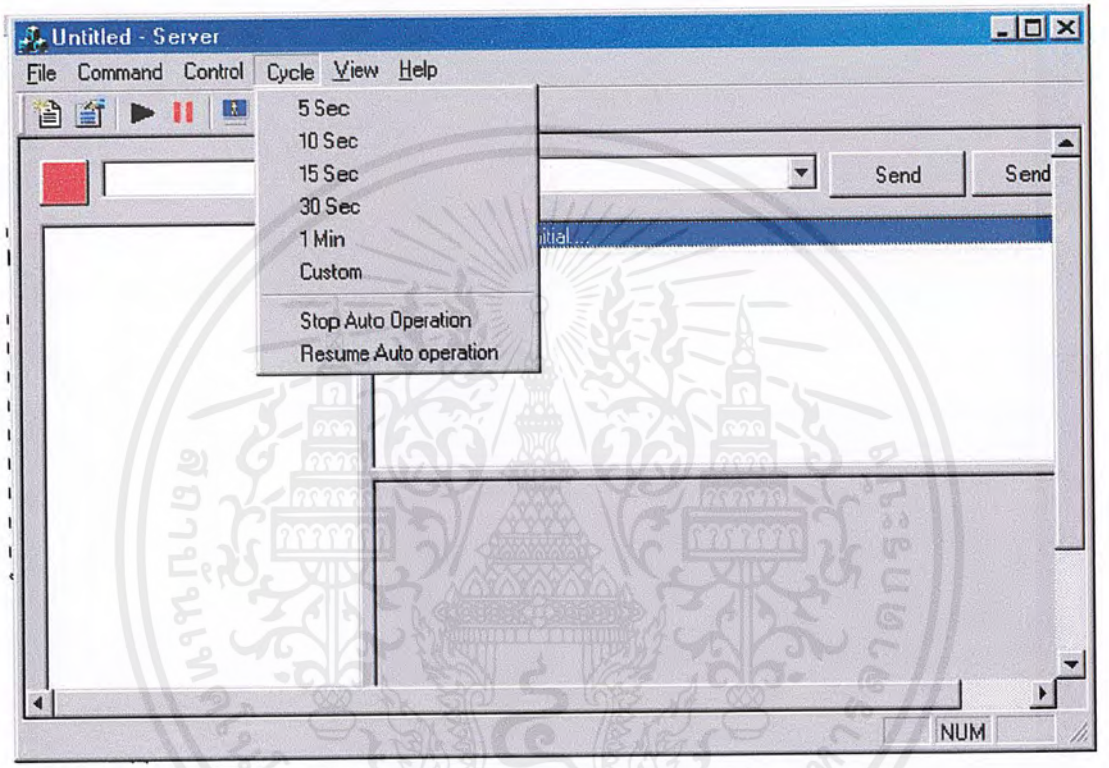
ลักษณะการทำงานของฟังก์ชันจะทำแบบวนรอบ เพื่อตรวจจับแอพพลิเคชันได้เรื่อย ๆ ให้ครบทุกเครื่องลูกข่ายที่อยู่ในระบบ ภายใต้ระยะเวลาหน่วงที่กำหนดไว้ และการทำงานสามารถที่จะถูกขัดจังหวะ (Interrupt) ได้ ถ้า Manual Operation ถูกเรียกใช้โดยจะส่งผ่านตัวแปร InterruptCycle โดยมีค่าดังนี้

- ถ้า InterruptCycle เป็น true ให้ Auto Operation หยุดทำงาน

- ถ้า InterruptCycle เป็น false Auto Operation จะทำงานปรกติ

ที่ทำเช่นนี้เนื่องจากว่า ในระยะเวลาหนึ่ง ๆ ข้อมูลจะส่งผ่านพอร์ตหนึ่ง ๆ ได้เพียงชุดเดียวเท่านั้น ดังนั้นถ้าAuto Operation กับ Manual Operation ส่งพร้อมๆ กันก็จะเกิดการผิดพลาดขึ้นทำให้ต้องมีการขัดจังหวะเกิดขึ้น

เราสามารถกำหนดระยะเวลาในการหน่วงได้จากเมนูดังนี้ (รูปที่ 5.15)



รูปที่5.15 เมนูแสดงระยะเวลาการหน่วง

2) การทำงานโดยปกติ (Manual Operation) หมายถึง ผู้ควบคุมระบบเป็นคนสั่งการทั้งหมด จึงกำหนดให้เป็นฟังก์ชันการทำงานที่มีลำดับความสำคัญ (priority) สูงสุด สามารถขัดจังหวะ Auto Operation ได้โดยจะมีการทำงาน คือ

- สามารถสั่งจับแอพพลิเคชัน
- สามารถสั่งจับหน้าจอ
- สั่งปิดระบบปฏิบัติการได้ (shutdown)
- ส่งข้อความได้โดยแจกแจงรายละเอียดต่อไป

2.1) การตรวจจับแอพพลิเคชัน (Capture Application) สามารถสั่งได้สองทางคือ การสั่งจากการเลือกเมนู และการเลือกจาก toolbar โดยมีหลักการทำงานเหมือนกันคือ เครื่องควบคุมจะทำการสร้างแมสเสจประเภท MSG_CAP_APP ขึ้นมาและทำการส่งไปยังเครื่องลูกข่าย ดัง โค้ด ต่อไปนี้

```
void CServerView::OnGetapp()
{
        // TODO: Add your command handler code here
        HTREEITEM htreeitem;
        if(m_tree.GetSelectedItem() != NULL)
        {
                htreeitem = m_tree.GetSelectedItem();
                CString ItemText(m_tree.GetItemText(htreeitem));
                UINT UserID;
                UserID = GetUserIDint(ItemText);
                CMessage getapp(MSG_CAP_APP);
                getapp.SetElementAt(0,GetHostIP());
                SendMsg(UserID,getapp);
                SetTopic("Capturing Application");
        }
        else MessageBox("Please selected CLIENT to procress...");
}
```

รูปที่ 5.16 แสดงฟังก์ชัน OnGetApp();

โดยโค้ดในส่วนอื่น ๆ นั้นมีหน้าที่ในการตรวจและกำหนดผู้ใช้ที่ถูกเลือกจาก TreeView ซึ่งจะได้มาเป็น IP ของเครื่องลูกข่ายนั้น ๆ แต่ที่เราต้องการคือ UserID เพราะฉะนั้นเราจึงทำการเรียกฟังก์ชัน GetUserID(int): เพื่อแปลงค่า IP เป็น UserID และทำการส่งไปโดยฟังก์ชัน SendMsg แต่ในกรณีที่ผู้ดูแลระบบไม่ได้เลือกผู้ใช้ โปรแกรมก็จะขึ้นข้อความเตือนให้เลือกผู้ใช้จาก TreeView ก่อนจึงจะสามารถทำการส่งได้

2.2) การจับหน้าจอ (Capture Monitor) คือการทำงานเพื่อนำหน้าจอที่กำลังแสดงอยู่ที่เครื่องลูกข่ายในขณะนั้น ๆ มาแสดงยังเครื่องควบคุม เราสามารถจะสั่งให้ดำเนินการแบบนี้ได้ 3 วิธี คือ

1) การเรียกจากเมนู

2) การเรียกจาก toolbar

3) การดับเบิลคลิกที่ IP ของเครื่องที่ต้องการ

โดยมีหลักการทำงานคือ เครื่องควบคุมจะทำการสร้างแมสเสจประเภท MSG_CAPTURE ขึ้นมาและส่งไปยัง เครื่องลูกข่ายดังโค้ดต่อไปนี้

```
void CServerView::OnCapture()
{
        // TODO: Add your command handler code here
        HTREEITEM htreeitem;
        if(m_tree.GetSelectedItem() != NULL)
        {
                htreeitem = m_tree.GetSelectedItem();
                CString ItemText(m_tree.GetItemText(htreeitem));
                UINT UserID;
                UserID = GetUserIDint(ItemText);
                CMessage capture(MSG_CAPTURE);
                capture.SetElementAt( 0,GetHostIP() );
                SendMsg(UserID,capture);
        bar.Create("Capture Screen", 40, 5000);
        for (int i = 0; i < 500; i++) {          // perform operation
                bar.StepIt();
        }
        }
        else MessageBox("Please selected CLIENT to procress...");

}
```

รูปที่ 5.17 แสดงฟังก์ชัน OnCapture();

โดยการทำงานในส่วนอื่น ๆ จะเหมือนกับการตรวจจับแอพพลิเคชัน เช่นการหา UserID จาก ผู้ใช้ที่ถูกเลือกใน TreeView เป็นต้น

2.3) การสั่งปิดระบบปฏิบัติการของเครื่องลูกข่าย

จุดประสงค์เพื่อสั่งหยุดการใช้งานของเครื่องลูกข่ายนั้น ๆ ลงโดยเครื่องควบคุมจะสร้างแมสเสจประเภท MSG_SHUTDOWN ขึ้นมาดังโค้ด

```
void CServerView::OnShutdown()
{
        HTREEITEM htreeitem;
        if(m_tree.GetSelectedItem() != NULL)
        {
                htreeitem = m_tree.GetSelectedItem();
                CString ItemText(m_tree.GetItemText(htreeitem));
                UINT UserID;
                UserID = GetUserIDint(ItemText);
                CMessage shutdown(MSG_SHUTDOWN);
                shutdown.SetElementAt(0,"Shuttingdown SYSTEM !!!");
                MessageBox("Shuttingdown SYSTEM !!!");
                SendMsg(UserID,shutdown);
        }
        else MessageBox("Please selected CLIENT to procress...");
}
```

รูปที่ 5.18 แสดงฟังก์ชัน OnShutdown();

การทำงานในส่วนอื่น ๆ ก็จะเหมือนกับการตรวจจับแอพพลิเคชัน แต่สำหรับการสั่งปิดระบบปฏิบัติการนี้ จะมีส่วนเพิ่มเติมขึ้นมาคือ เราสามารถสั่งปิดเครื่องลูกข่ายทุก ๆ เครื่องพร้อม ๆ กันโดย

ทำการส่ง MSG_SHUTDOWN ครั้งเดียวได้โดยเรียกใช้ฟังก์ชัน SendMessageToAll() ซึ่งจะทำให้เราไม่จำเป็นที่จะต้องมีอัลกอริทึมในการหาผู้ใช้ที่จะเลือกเพื่อสั่งปิด เนื่องจากเราสั่งปิดทุกเครื่อง ดังโค้ด

```
void CServerView::OnShutdownall()
{
        // TODO: Add your command handler code here
        CMessage shutdownall(MSG_SHUTDOWN);
        shutdownall.SetElementAt(0,"Shuttingdown SYSTEM !!!");
        MessageBox("Shuttingdown All SYSTEM !!!");
        SendMessageToAll(shutdownall);
}
```

รูปที่ 5.19 แสดงฟังก์ชัน **OnShutdownall();**

2.4) การส่งข้อความไปยังเครื่องลูกข่าย

ความสามารถโดยพื้นฐานของโปรแกรมที่ทำงานในลักษณะไคลเอนต์และเซิร์ฟเวอร์นี้ จะมีความสามารถในการส่งข้อความถึงกันได้ เพราะฉะนั้นความสามารถของโปรแกรมในส่วนนี้จึงทำขึ้นเพื่อให้โปรแกรม PCLC-Server สามารถที่จะส่งข้อความไปยังเครื่องลูกข่ายเครื่องใด ๆ ก็ได้ที่ต่ออยู่ภายในระบบ โดยโปรแกรม PCLC-Server ที่เครื่องผู้ควบคุมจะทำการสร้างแมสเสจประเภท MAG_RECEIVE_TEXT ขึ้นมาดังโค้ด

```
void CServerView::OnSend()
{
        HTREEITEM htreeitem;
        if(m_tree.GetSelectedItem() != NULL)
        {
                htreeitem = m_tree.GetSelectedItem();
                CString ItemText(m_tree.GetItemText(htreeitem));
                UINT UserID;
                UserID = GetUserIDint(ItemText);
                CMessage send(MSG_RECEIVE_TEXT);
                UpdateData(TRUE);
                if (m_command == "")
                        MessageBox("Message is empty");
                else
                {
                        m_command_control.AddString(m_command);
                        send.SetElementAt(0,m_command);
                        SendMsg(UserID,send);
                }
        }
        else MessageBox("Please selected CLIENT to procress...");
}
```

รูปที่ 5.20 แสดงฟังก์ชัน **OnSend();**

โดยมีการทำงานในส่วน get user เหมือนตรวจจับแอพพลิเคชันและจะนำข้อความที่อยู่ในลิสต์บ็อกซ์ ส่งไปให้ผู้ใช้ที่ได้ทำการเลือกไว้ และมีความสามารถที่จะส่งข้อความเดียวกันไปยังเครื่องลูกข่ายทุก ๆ เครื่องได้ ด้วยเหมือนกับการส่ง ShutdownAll โดยการใช้ฟังก์ชันในการส่งแมสเสจไปสู่ทุก ๆ เครื่องลูกข่ายคือ SendMessageToAll() ดังโค้ด

```
void CServerView::OnSendall()
{
        // TODO: Add your control notification handler code here
        CMessage SendMsgToAll(MSG_RECEIVE_TEXT);
        UpdateData(TRUE);
        if (m_command == "")
                MessageBox("Message is empty");
        else
        {
                SendMsgToAll.SetElementAt(0,m_command);
                SendMessageToAll(SendMsgToAll);
        }
}
```

รูปที่ 5.21 แสดงฟังก์ชัน **OnSendall();**

5.3 ความสามารถพิเศษอื่น ๆ ของโปรแกรม

โดยปรกติแล้ว Auto-Operation จะทำงานแยกอิสระและเป็นอัตโนมัติคือจะทำงานตามที่ได้กำหนดไว้ตลอดเวลาและถูกต้องแต่ในกรณีที่เราต้องการจะสั่งหยุดการทำงานแบบ Auto-Operation นี้ โปรแกรม Client จะมีฟังก์ชันที่ทำหน้าที่จัดการในส่วนนี้อยู่ 2 ฟังก์ชันด้วยกัน คือ

1. เป็นการสั่งให้ Auto-Operation หยุดทำงานโดยการกำหนดให้ InterruptCycle = true
2. เป็นการสั่งให้ Auto-Operation ทำงานตามปกติโดยกำหนดให้ InterruptCycle = false

การกำหนดค่าหน่วงเวลาการทำงานของ Auto-Operation

เราสามารถกำหนดระยะเวลาในการหน่วงเวลาการทำงานของ Auto-Operation ได้โดยการเลือกจากเมนู ซึ่งจะมีค่าที่สามารถเลือกใช้และกำหนดได้ดังนี้

1). 5 sec กำหนดให้หน่วงเวลา 5 วินาทีดังโค้ด

```
void CServerView::OnCycle5sec()
{
        CycleTime = 5;
}
```

รูปที่ 5.22 แสดงฟังก์ชัน **OnCycle5sec();**

2). 10 sec กำหนดให้หน่วงเวลา 10 วินาทีดังโค้ด

```
void CServerView::OnCycle10sec()
{
        CycleTime = 10;
}
```

รูปที่ 5.23 แสดงฟังก์ชัน **OnCycle10sec();**

3). 15 sec กำหนดให้หน่วงเวลา 15 วินาทีดังโค้ด

```
void CServerView::OnCycle15sec()
{
        CycleTime = 15;
}
```

รูปที่ 5.24 แสดงฟังก์ชัน **OnCycle15sec();**

4). 30 sec กำหนดให้หน่วงเวลา 30 วินาทีดังโค้ด

```
void CServerView::OnCycle30sec()
{
        CycleTime = 30;
}
```

รูปที่ 5.25 แสดงฟังก์ชัน **OnCycle30sec();**

5) 1 min กำหนดให้หน่วงเวลา 1 นาทีดังโค้ด

```
void CServerView::OnCycle1min()
{
        CycleTime = 60;
}
```

รูปที่ 5.26 แสดงฟังก์ชัน **OnCycle1min();**

6). Custom กำหนดค่าการหน่วงเวลาตามที่ผู้ควบคุมระบบต้องการ มีหน่วยเป็นนาที

หมายเหตุ ค่าหน่วงเวลายิ่งน้อยจะยิ่งทำให้ระบบเครือข่ายทำงานได้ช้าลง เนื่องจากข้อมูลในระบบจะมีมากขึ้น

5.4 ฟังก์ชันอื่น ๆ ที่เกี่ยวข้องกับการทำงาน

การทำงานในส่วนของ UserList

1. AddUser เป็นฟังก์ชันที่ถูกเรียกใช้เมื่อมีผู้ใช้ติดต่อเข้ามาในระบบมีหน้าที่ในการเก็บค่า UserID ให้ตรงกับ IP ของเครื่องลูกข่ายนั้น ดัง โค้ด

```
void CServerView::AddUser(UINT id,const CString &user)
{
        UserList[id] = user;
        LastUserID++;
}
```

รูปที่ 5.27 แสดงฟังก์ชัน **AddUser();**

บทที่ 6
การออกแบบโปรแกรม PCLC-Client

การทำงานทุกอย่างของเครื่องลูกข่ายนั้นจะไปตามเหตุการณ์ของเครื่องควบคุม หรือพูดอีกนัยหนึ่งก็ คือ เครื่องลูกข่ายจะไม่มีการทำงานใด ๆ เลย ถ้าไม่ได้รับคำสั่งหรือแมสเสจมาจากเครื่องควบคุม ตามขั้นตอนต่อไปนี้คือ

6.1 การโหลดไฟล์ Configuration

โดยในขั้นตอนแรกของการทำงาน จะต้องทำการโหลดเอารายละเอียดที่จำเป็นในการที่จะต้องติดต่อไปหาเครื่องควบคุม 2 อย่างคือ

- หมายเลข IP ของเครื่องควบคุม
- พอร์ตที่ใช้ในการติดต่อกับเครื่องควบคุม

โดยรายละเอียดทั้งสองนี้จะเก็บอยู่ในไฟล์ **CLIENT.INI** ข้อมูลที่จัดเก็บอยู่ภายในไฟล์ประเภท INI นั้นจะเก็บในลักษณะ **แอททริบิวต์ = ค่าของแอททริบิวต์** ซึ่งค่าที่เก็บอยู่ในไฟล์ CLIENT.INI คือ HostIP = 127.0.0.1 และ HostPort = 5000 ในการโหลดเอาหมายเลข IP ของเครื่องควบคุมมาจากไฟล์ CLIENT.INI นั้น จะใช้ **ฟังก์ชัน GetProfileString ("Initial","HostIP","127.0.0.1");** ดังรูปที่ 6.1

ซึ่งจะโหลดค่าของแอททริบิวต์จากบรรทัดที่ เริ่มต้นด้วยค่าแอททริบิวต์ HostIP แล้วนำค่าของแอททริบิวต์ที่ได้นั้นมาเก็บลงในตัวแปรแบบสตริง (HostIP) และในการโหลดเอาหมายเลขพอร์ตก็เช่นเดียวกันแต่จะใช้ฟังก์ชัน GetProfileInt ("Initial" , "HostPort" , 5000) แทนเพื่อนำค่าของแอททริบิวต์ที่ได้นั้นมาเก็บลงตัวแปร (HostPort) ในลักษณะของจำนวนเต็มแทน ซึ่งสามารถแสดงการทำงานได้ดังนี้

```
HostIP = ((CWinApp*)AfxGetApp())->GetProfileString("Initial","HostIP","127.0.0.1");
HostPort = ((CWinApp*)AfxGetApp())->GetProfileInt("Initial","HostPort",5000);
```

รูปที่ 6.1 แสดงฟังก์ชันการโหลดค่า HostIP และ HostPort

6.2 การตรวจสอบหมายเลข IP

เมื่อโหลดค่าทั้ง 2 ค่านี้เรียบร้อยแล้ว จะต้องทำการตรวจสอบว่า หมายเลข IP ที่โหลดขึ้นมาได้นั้นมีความถูกต้องหรือไม่ ถ้าไม่ถูกต้องก็จะมีการแสดงกล่องข้อความเพื่อแสดงข้อความแจ้งให้กับเครื่องลูกข่ายทราบดังรูปที่ 6.2

```
if (!CIp::IsValid(HostIP))
MessageBox("Invalid IP address", "Error", MB_OK | MB_ICONSTOP);
```

รูปที่ 6.2 แสดงกล่องข้อความเพื่อแจ้งความประสงค์ให้กับเครื่องลูกข่าย

6.3 เซ็ตค่า HostIP

ต่อไปก็จะทำการเซ็ตค่า HostIP เพื่อเตรียมพร้อมที่จะเรียกฟังก์ชันเปิดการเชื่อมต่อไปยังเครื่องควบคุม ดังรูปที่ 6.3

```
CIp ip;
ip.SetIp(HostIP);
```

รูปที่ 6.3 แสดงการเซ็ตค่า HostIP

6.4 เปิดการเชื่อมต่อไปยังเครื่องควบคุม

ในการเปิดการเชื่อมต่อไปยังเครื่องควบคุม หากว่าเครื่องควบคุมยังไม่ได้ทำงานอยู่ โปรแกรม PCLC-Client จะทำการ reconnect เรื่อย ๆ จนกว่า โปรแกรม PCLC-Server จะทำงาน ดังนั้นจึงไม่มีความจำเป็นที่โปรแกรม PCLC-Server จะต้องทำงานก่อนโปรแกรม PCLC-Client โดยในการเปิดการเชื่อมต่อไปยังเครื่องควบคุมจะมีการกำหนดรายละเอียดต่าง ๆ ดังนี้

```
if (OpenConnection(ip, HostPort))
        {
                MessageBox("* You are connected to the server at" + HostIP,"Connecting to SERVER");
                CMessage msg(MSG_SEND_TEXT);
                msg.SetElementAt(0,GetClientIP());
                file(filename,CFile::modeCreate|CFile::modeWrite);
                CString temp;
                char tmpbuf[128];
              _tzset();
              _strtime( tmpbuf );
                temp.Format("Connected at: %s",tmpbuf);
                msg.SetElementAt(1,temp);
                msg.SetElementAt(2,"Port:5000");
                DWORD    NTVersion;
                CString IsNT;
                NTVersion = GetVersion();
                if( NTVersion >= 0x80000000 ) {
                        IsNT = "Windows 9x";
                } else {
                        IsNT = "Windows NT";
                }
                temp.Format("OS: %s",IsNT);
                msg.SetElementAt(3,temp);
                SendMsg(msg);
        }
        else
        {


                MessageBox("Cannot connect to the server at " + HostIP, "Error", MB_OK | MB_ICONSTOP);
                ClientExitFlag = true;                  //checking when can't connected to server
        }
```

รูปที่ 6.4 เปิดการเชื่อมต่อไปยังเครื่องควบคุมโดยมีการกำหนดรายละเอียดต่าง ๆ

6.5 การดักฟังแมสเสจ

เมื่อเครื่องลูกข่ายสามารถเปิดการเชื่อมต่อไปยังเครื่องควบคุมได้แล้ว ณ ขณะนี้เครื่องควบคุมและเครื่องลูกข่ายก็สามารถที่จะทำการติดต่อสื่อสารกันได้แล้ว

สิ่งที่เครื่องลูกข่ายต้องทำในอันดับต่อไปก็คือ การดักฟังแมสเสจประเภทต่าง ๆ ที่จะถูกส่งมาจากเครื่องควบคุม จุดประสงค์ก็คือต้องการให้เครื่องลูกข่ายมีการทำงานตามที่เครื่องควบคุมส่งมา โดยในการดักจับแมสเสจประเภทต่าง ๆ ที่มาจากเครื่องควบคุมนั้นจะใช้ **ฟังก์ชัน void OnMessage (CMessage &msg);** คอยดักจับ ซึ่งแมสเสจแต่ละอันนั้นจะบอกจุดประสงค์ของเครื่องควบคุมว่าต้องการให้เครื่องลูกข่ายมีการทำงานอะไร แล้วเครื่องลูกข่ายก็มีหน้าที่ตอบสนองต่อแมสเสจที่เครื่องควบคุมส่งมา โดยแมสเสจต่าง ๆ ที่ถูกส่งมาจากเครื่องควบคุมมีดังนี้คือ

1) ***MSG_CAPTURE*** จะถูกส่งมาจากเครื่องควบคุมเมื่อเครื่องควบคุมต้องการที่จะจับภาพหน้าจอของเครื่องลูกข่าย ซึ่งการทำงานเมื่อได้รับแมสเสจนี้จะแบ่งออกได้เป็น 3 ส่วน (รูปที่ 6.5) ก็คือ

1.1) ทำการโหลดเอาหมายเลข IP ของเครื่องขึ้นมา เพื่อนำไปใช้เป็นชื่อไฟล์ที่ใช้ในการจับภาพหน้าจอ หมายความว่า ถ้าเราต้องการจับภาพหน้าจอของเครื่องลูกข่ายเครื่องใด ก็ให้ใช้ หมายเลข IP ของเครื่องนั้นมาใช้เป็นชื่อไฟล์ โดยนามสกุลของไฟล์จะเป็นประเภท .BMP

1.2) จากนั้นส่งชื่อไฟล์ที่โหลดมาได้ เพื่อใช้เป็นพารามิเตอร์ส่งไปยังฟังก์ชันการจับภาพหน้าจอ ซึ่งรายละเอียดของฟังก์ชันนี้สามารถอ่านรายละเอียดได้ที่ฟังก์ชันการทำงานของโปรแกรม

1.3) เมื่อได้ไฟล์ภาพหน้าจอแล้วก็จะทำการบีบอัดไฟล์เป็นสกุล ZIP ก่อนส่งไปยังเครื่องควบคุมเพื่อทำการแสดงผลต่อไปซึ่งในการส่งไฟล์ไปแสดงผลยังเครื่องควบคุมนั้น สิ่งที่จำเป็นต้องส่งไปนั้นมี 3 ส่วนคือ

- ชื่อไฟล์
- ขนาดของไฟล์
- ข้อมูลภาพ

```
case MSG_CAPTURE:
{
        CString filename(GetClientIP() + ".BMP");
        if ( CaptureToFile(filename) )
        SendFile(filename);
}
break;
```

รูปที่ 6.5 แสดง MSG_CAPTURE

ซึ่งในการส่งข้อมูลทั้ง 3 ส่วนนี้จะใช้ฟังก์ชัน SendFile ดังนี้

```
void CMainFrame::SendFile(CString FileName)
{
	CFile myFile;
	if (!myFile.Open(FileName, CFile::modeRead | CFile::typeBinary) )
	{
		MessageBox("Can't open file...");
		return;
	}

	int myFileLength = myFile.GetLength(); // Going to send the correct File Size
	CMessage msg_fname(MSG_FILE_NAME);
	msg_fname.SetElementAt(0,FileName);
	SendMsg(msg_fname);                                    //send file_name

	CMessage msg_length(MSG_DATA_LENGTH);
	char buffer[20];
	_itoa( myFileLength, buffer, 10 );
	msg_length.SetElementAt(0,buffer);
	SendMsg(msg_length);                                   //send byte_length of file
	byte* data = new byte[myFileLength];
	myFile.Read(data, myFileLength);
	CMessage msg_data(MSG_SEND_FILE);
	msg_data.SetElementAt(0,data,myFileLength);
	SendMsg(msg_data);                                     //send file_data
	myFile.Close();
	delete data;
}
```

รูปที่ 6.6 แสดงฟังก์ชัน **SendFile();**

2) MSG_RECEIVE_TEXT แมสเสจนี้จะถูกส่งมาจากเครื่องควบคุมเมื่อเครื่องควบคุมต้องการที่จะส่งข้อความติดต่อมายังเครื่องลูกข่าย โดยเมื่อเครื่องลูกข่ายได้รับแมสเสจนี้จากเครื่องควบคุมแล้วก็ จะทำการอ่านข้อความที่อยู่ในช่องข้อความซึ่งถูกส่งมาพร้อมกับแมสเสจ แล้วใช้กล่องข้อความแสดงข้อความนั้นออกมาดังรูปที่ 6.7

```
case MSG_RECEIVE_TEXT:
{
	CString text;
	msg.GetElementAt(0, text);
	MessageBox(text,"Message from SERVER at " + HostIP);
}
break;
```

รูปที่ 6.7 แสดง MSG_RECEIVE_TEXT

3) MSG_CAP_APP แมสเสจนี้จะถูกส่งมาจากเครื่องควบคุมเมื่อเครื่องควบคุมต้องการที่จะตรวจสอบการใช้งานแอพพลิเคชันของเครื่องลูกข่ายโดยทำการเรียกไฟล์ ps.exe เพื่อเรียกโปรแกรมขึ้นมาทำการจับโปรเซสการทำงานของเครื่องลูกข่าย จากนั้นก็ทำการบันทึกโปรเซสที่ตรวจสอบได้เอาไว้ในไฟล์ ps.txt จากนั้นจะเปิดไฟล์ดังกล่าวขึ้นมาเพื่ออ่าน และทำการอ่านขึ้นมาทีละบรรทัดจนจบไฟล์ และส่งกลับไปแสดงผลยังเครื่องควบคุม

```
case MSG_CAP_APP:
{
        CString CommandLine("command.com /c ps.exe > ps.txt");
        WinExec(CommandLine,SW_SHOWMINNOACTIVE);
        CStdioFile s("ps.txt",CFile::modeRead);
        CString buff;
        CMessage app(MSG_CAP_APP);
        int index;
        index=0;
        while(!feof(s.m_pStream))
        {
                s.ReadString(buff);
                app.SetElementAt(index,buff);
                index += 1;
        }
        SendMsg(app);
}
break;
```

รูปที่ 6.8 แสดง MSG_CAP_APP

4) MSG_SHUTDOWN แมสเสจนี้จะถูกส่งมาจากเครื่องควบคุมเมื่อเครื่องควบคุมต้องการที่จะปิดการทำงานของเครื่องลูกข่าย โดยใช้ฟังก์ชัน **ExitWindowsEx** และสามารถเลือกผ่านพารามิเตอร์เข้าไปได้ 2 ตัวซึ่งตัวแรกจะเป็นพารามิเตอร์ที่ใช้ควบคุมในกรณีที่ระบบปฏิบัติการที่ใช้เป็น Windows 9x และพารามิเตอร์ตัวที่สองจะใช้ควบคุมในกรณีที่ระบบปฏิบัติการที่ใช้เป็น Windows NT ซึ่งตามระบบนั้นเครื่องลูกข่ายจะใช้ระบบปฏิบัติการ Windows 9x ดังนั้นจึงใช้ EWX_SHUTDOWN เป็นพารามิเตอร์ตัวแรกซึ่งใช้ในการปิดการทำงานของวินโดวส์ดังนี้

```
case MSG_SHUTDOWN:
{
        CString text;
        msg.GetElementAt(0,text);
        MessageBox(text);
        ExitWindowsEx(EWX_SHUTDOWN,NULL);
}
break;
```

รูปที่ 6.9 แสดง MSG_SHUTDOWN

6.6 กรณีที่เกิดการดักฟังผิดพลาด

ต่อจากนั้นก็จะเป็นการดักฟังในกรณีที่เกิดความผิดพลาดในการเชื่อมต่อ ซึ่งความผิดพลาดที่เกิดขึ้นนี้เป็นความผิดพลาดที่ไม่สามารถระบุความผิดพลาดลงไปได้ ซึ่งถ้าเกิดกรณีนี้ขึ้นก็จะมีการตัดการเชื่อมต่อระหว่าง 2 ฝั่ง และทำการแจ้งไปยังเครื่องลูกข่ายว่าได้ทำการตัดการเชื่อมต่อแล้ว

```
void CMainFrame::OnDisconnect(EDisconnectionType reason)
{
        MessageBox("Disconnecting...");
}
```

รูปที่ 6.10 แสดงการแจ้งไปยังเครื่องลูกข่ายว่าตัดการสื่อสารแล้ว

ส่วนต่อไปก็คือส่วนสุดท้ายของการเชื่อมต่อ เมื่อเครื่องลูกข่ายถูกสั่งให้ปิดการทำงานไม่ว่าจะเป็นการสั่ง shutdown หรือว่า ปิดระบบปฏิบัติการ สิ่งที่จะต้องทำก็คือ

- ปิดการเชื่อมต่อ
- ส่งแมสเสจ ***MSG_USER_QUIT*** ตอบกลับไปยังเครื่องควบคุม

6.7 กรณีพิเศษ

ในกรณีที่มีผู้ที่พยายามจะปิดการทำงานของโปรแกรมควบคุมทางฝั่งเครื่องลูกข่ายลง เพื่อที่จะให้โปรแกรมยังคงทำงานอยู่ได้ดีโดยตลอด เราจะใช้อัลกอริทึมพิเศษที่จะถูกเรียกใช้งานเมื่อใดก็ตามที่มีผู้พยายามที่จะปิดการทำงานของโปรแกรม โดยมันจะทำการเรียกโปรแกรมขึ้นมาใช้งานใหม่อีกครั้งโดยอัตโนมัติ ซึ่งสามารถแสดงได้ดังนี้

```
int CSystemTrayApp::ExitInstance()
{
        // TODO: Add your specialized code here and/or call the base class
        //This Method Called when exiting program...
        ((CMainFrame*)AfxGetApp())->OnClientExit();
        if(ExitFlag)
        {
                ((CMainFrame*)AfxGetApp())->MessageBox("Abnormal exiting...");
                WinExec("client.exe",SW_SHOW);
        }
        return CWinApp::ExitInstance();
}
```

รูปที่ 6.11 แสดงอัลกอริทึมพิเศษที่ใช้ในกรณีพิเศษ

บทที่ 7
ผลการดำเนินงาน

7.1 การพัฒนาโปรแกรม

7.1.1 เครื่องมือที่ใช้ในการพัฒนาโปรแกรม

1) ภาษาที่ใช้ในการพัฒนาโปรแกรม ได้แก่ ภาษา Visual C++

2) ระบบปฏิบัติการที่ใช้ได้แก่ Windows 98 หรือ Windows 95 และ Windows NT เซิร์ฟเวอร์ 4.0 Thai Enabled

3) โปรโตคอลที่ใช้ คือ โปรโตคอล TCP/IP

7.1.2 ผลการดำเนินงาน

โปรแกรมที่ได้พัฒนาแบ่งเป็น 2 ส่วนคือ โปรแกรมเครื่องควบคุมและโปรแกรมเครื่องลูกข่าย

1) โปรแกรมเครื่องควบคุม ซึ่งมีความสามารถในการควบคุมการทำงานของเครื่องลูกข่าย ทำการพัฒนาสำเร็จโดยใช้ชื่อ ***PCLC – Server*** ระบบปฏิบัติการที่เครื่องควบคุมได้แก่ Windows 95 หรือ Windows 98 และ Windows NT เซิร์ฟเวอร์ 4.0 หรือดีกว่า

2) โปรแกรมเครื่องลูกข่าย ซึ่งมีความสามารถในการรับคำสั่งจากเครื่องควบคุมมาประมวลผล แล้วดำเนินการตามคำสั่งนั้น ทำการพัฒนาสำเร็จโดยใช้ชื่อ ***PCLC – Client*** ระบบปฏิบัติการที่เครื่องลูกข่ายใช้ได้แก่ Windows 95 หรือ Windows 98

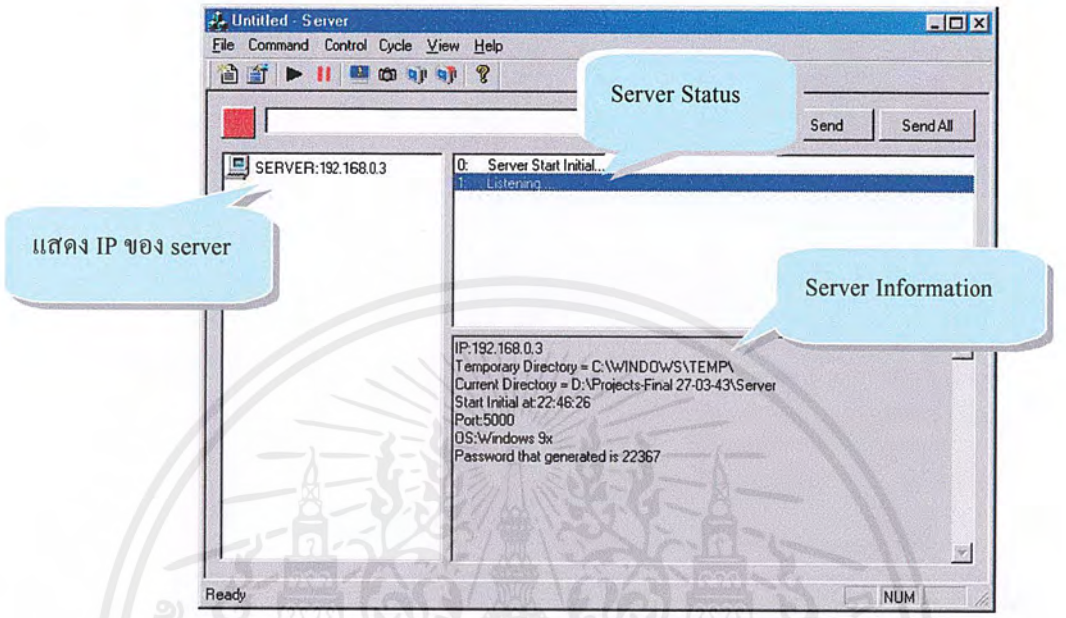
7.1.3 ข้อบังคับ

เนื่องจากไฟล์ภาพหน้าจอที่ได้มีขนาดใหญ่ ทำให้การส่งไฟล์ภาพหน้าจอผ่านเครือข่าย เป็นไปอย่างล่าช้า ดังนั้นก่อนที่จะทำการส่งไฟล์จึงต้องมีการบีบอัดข้อมูลด้วยโปรแกรม pkzip.exe ก่อน จากนั้นจึงใช้โปรแกรม pkunzip.exe เพื่อทำการคลายการบีบอัดข้อมูล

7.2 การทำงานหลักของโปรแกรมเครื่องควบคุม

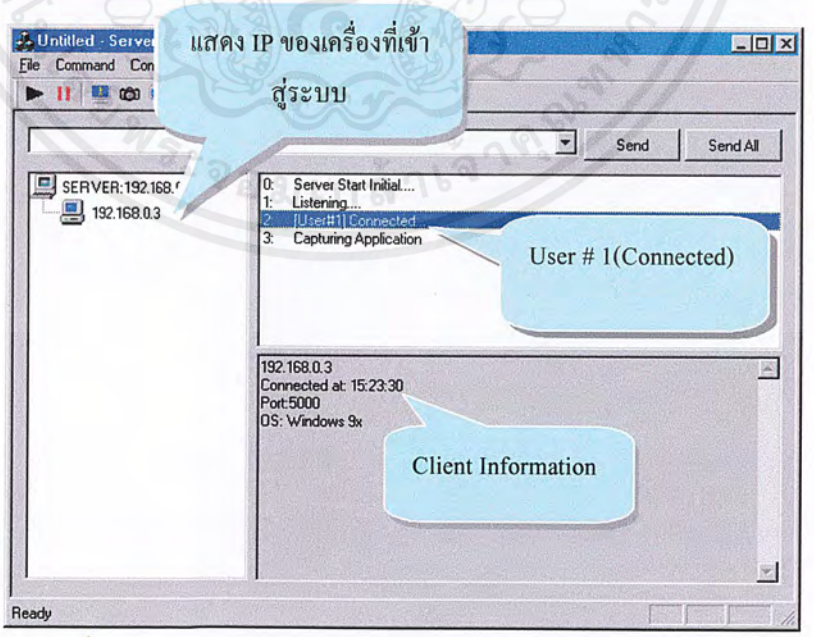
มีการทำงานเป็นขั้นตอนดังต่อไปนี้

7.2.1 ทำการเปิดโปรแกรมเซิร์ฟเวอร์ขึ้นมาเพื่อรอการติดต่อจากเครื่องลูกข่ายเข้าสู่ระบบ



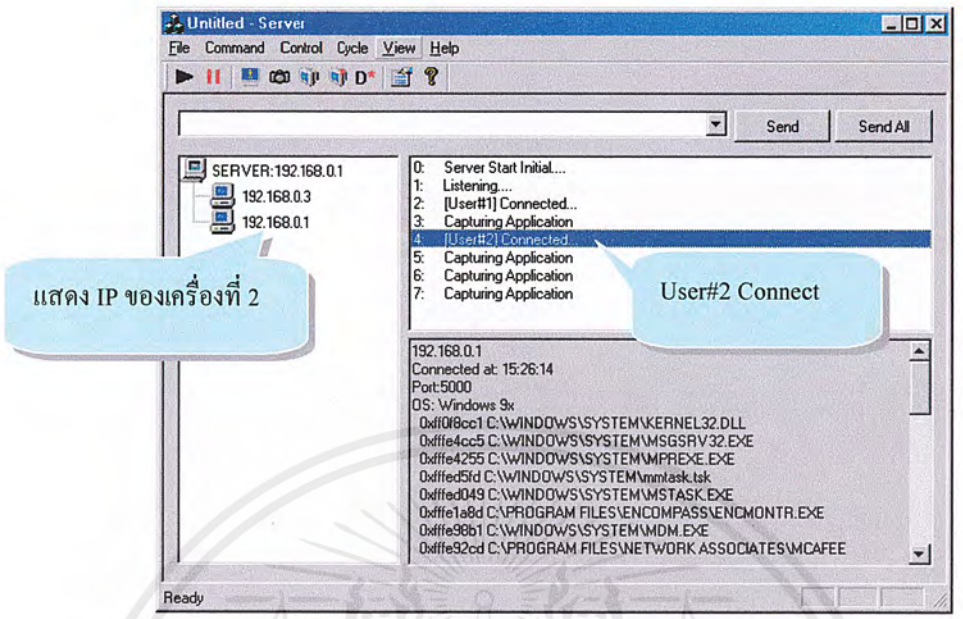
รูปที่ 7.1 แสดงส่วนการติดต่อกับโปรแกรม Server

7.2.2 เมื่อเครื่องลูกข่ายเครื่องแรกเข้าสู่ระบบ



รูปที่ 7.2 แสดงการเข้าสู่ระบบของเครื่องลูกข่ายเครื่องแรก

7.2.3 เมื่อเครื่องลูกข่ายเครื่องที่ 2 เข้าสู่ระบบ

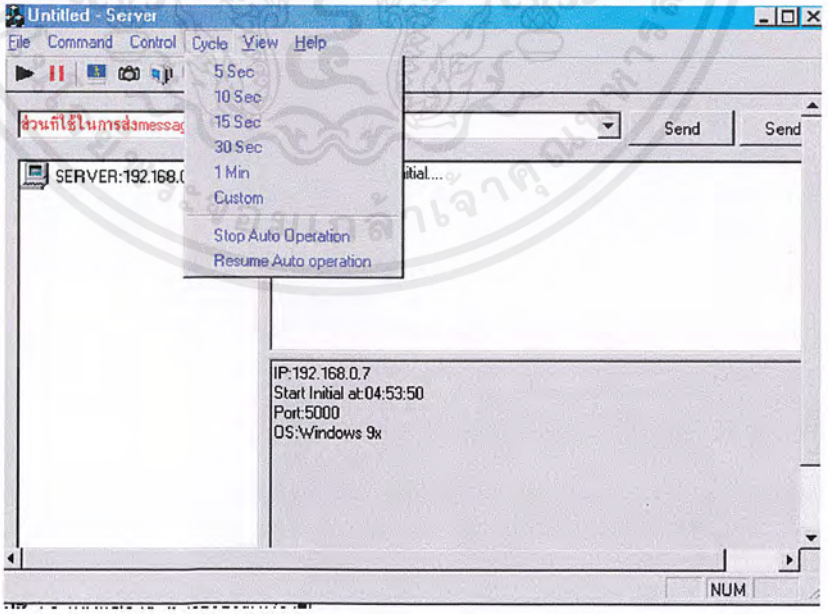


รูปที่ 7.3 แสดงการเข้าสู่ระบบของเครื่องลูกข่ายเครื่องที่ 2

7.2.4 เครื่องควบคุมสั่งให้ตรวจจับแอพลิเคชันที่เครื่องลูกข่ายสามารถทำได้ 2 แบบคือ

แบบที่ 1 การตรวจจับแอพพลิเคชันแบบ Auto Operation ซึ่งมีขั้นตอนดังนี้

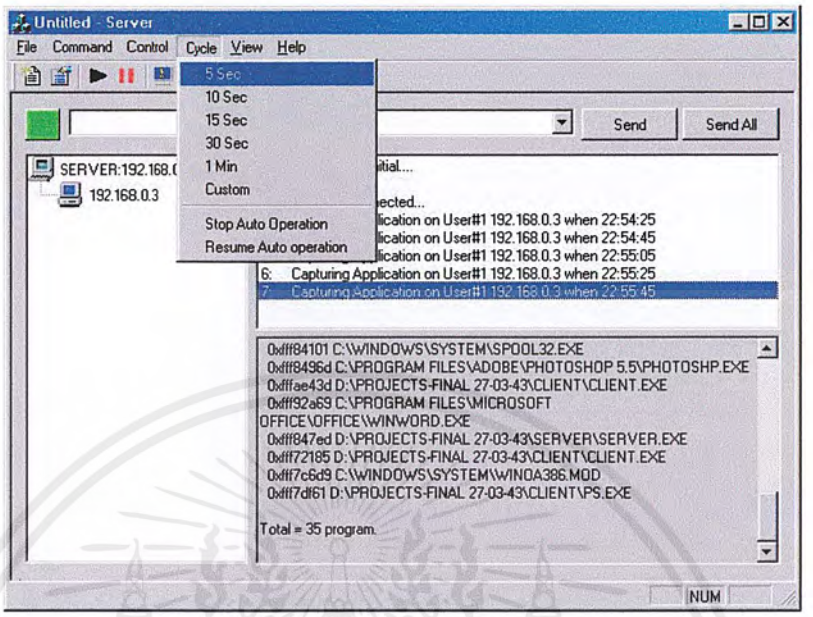
ขั้นตอนที่ 1 คลิกที่ Cycle บนเมนูบาร์ เพื่อเลือกระยะเวลาการหน่วงที่ต้องการ



รูปที่ 7.4 เมนูแสดงระยะเวลาการหน่วง

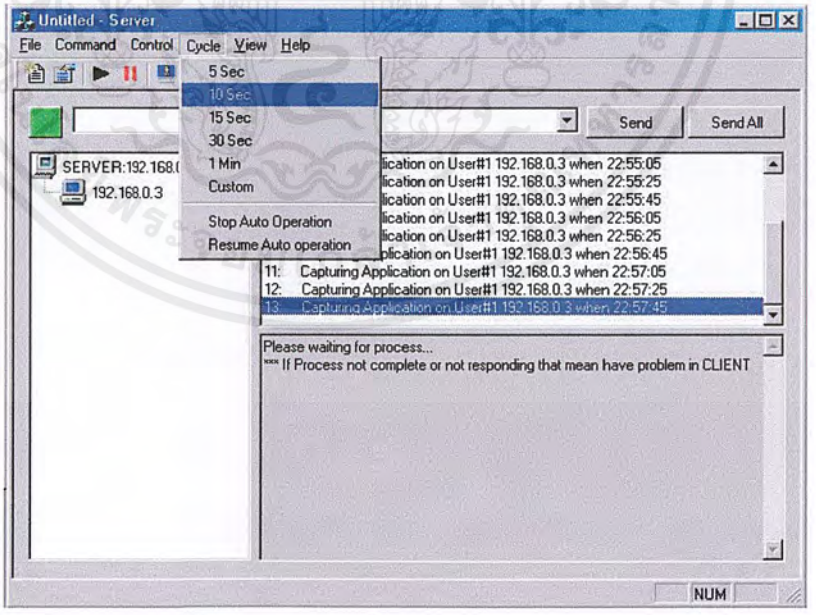
ขั้นตอนที่ 2 เลือกระยะเวลาหน่วงที่ต้องการซึ่งมีตัวเลือกดังนี้

1) 5 วินาที



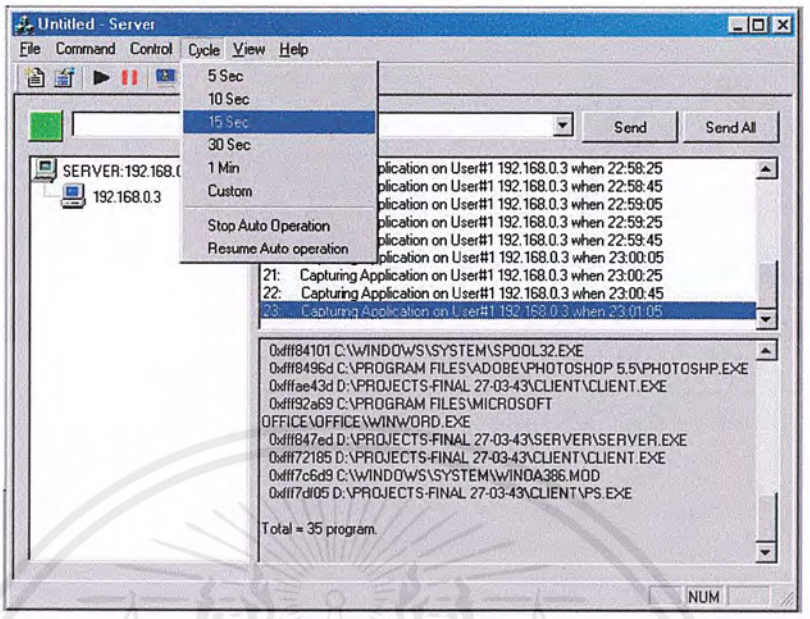
รูปที่ 7.5 แสดงระยะหน่วง 5 วินาที

2) 10 วินาที



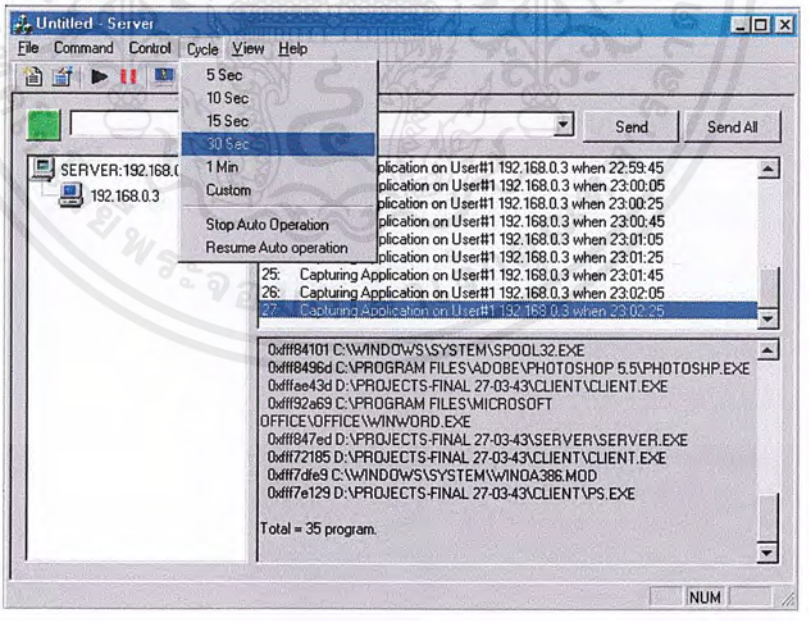
รูปที่ 7.6 แสดงระยะเวลาหน่วง 10 วินาที

3) 15 วินาที



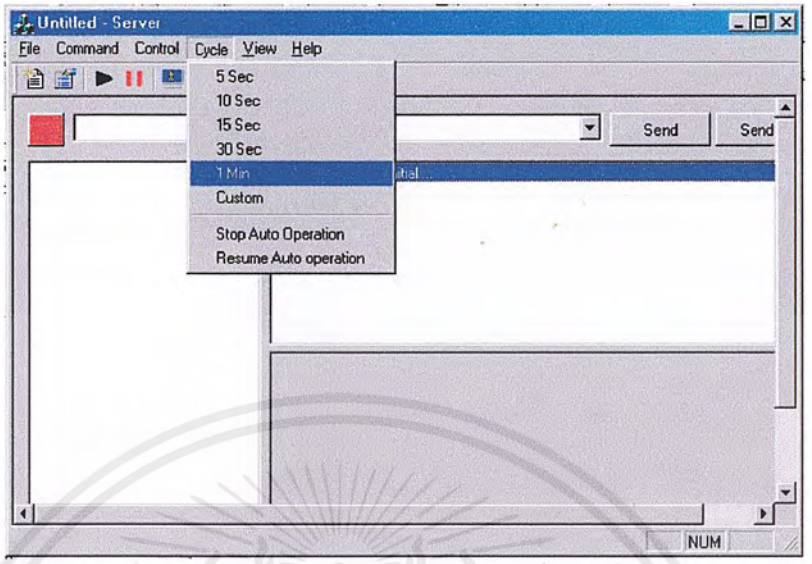
รูปที่ 7.7 แสดงระยะเวลาการหน่วง 15 วินาที

4) 30 วินาที



รูปที่ 7.8 แสดงระยะเวลาการหน่วง 30 วินาที

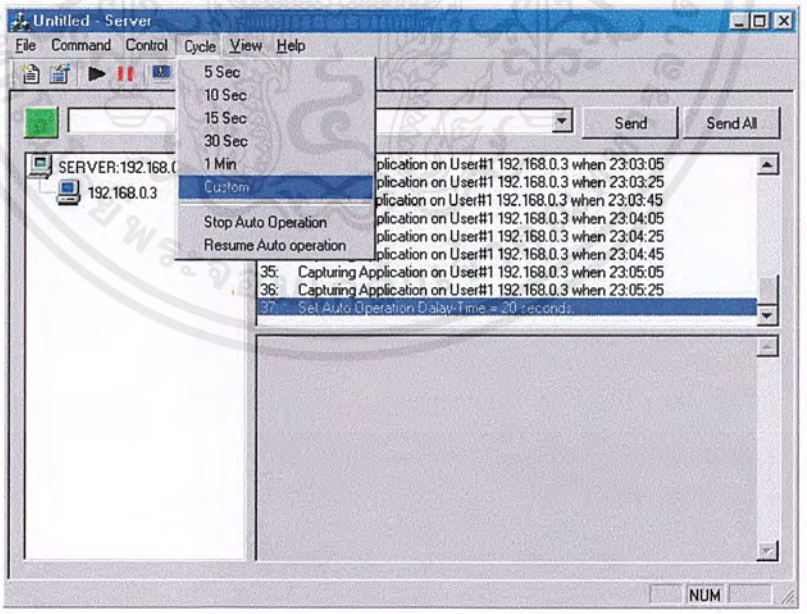
5) 1 นาที



รูปที่ 7.9 แสดงระยะเวลาการหน่วง 1 นาที

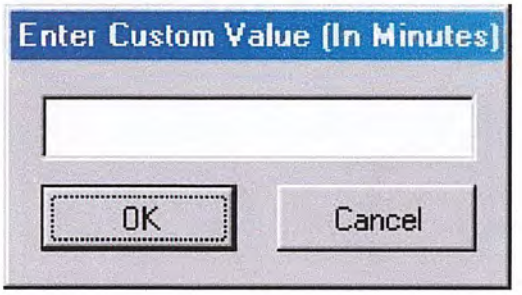
6) แบบ Custom มี 2 ขั้นตอนคือ

ขั้นตอนที่ 1 คลิกเลือก Custom



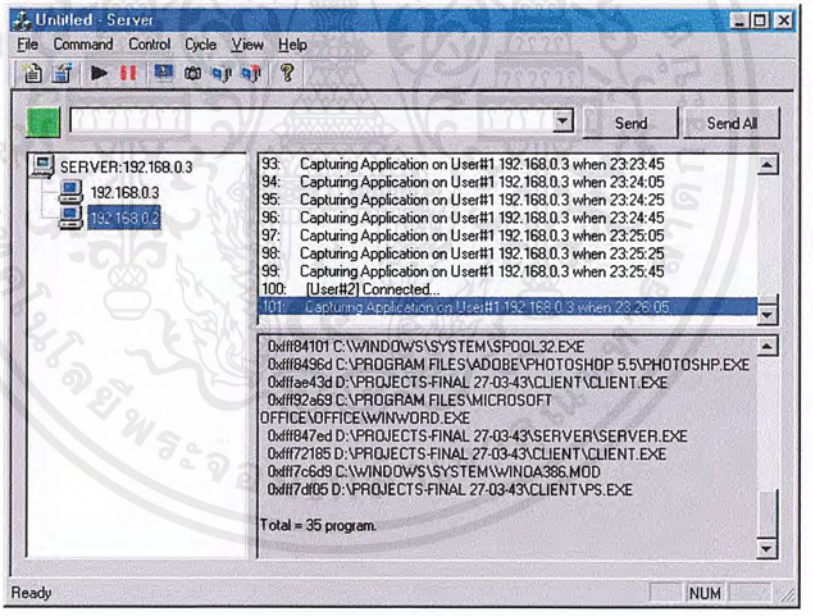
รูปที่ 7.10 แสดงขั้นตอนการหน่วงระยะเวลาแบบ Custom

ขั้นตอนที่ 2 ใส่จำนวนระยะเวลาการหน่วงที่ต้องการ



รูปที่ 7.11 แสดงกล่องข้อความเพื่อให้กรอกระยะเวลาการหน่วงที่ต้องการซึ่งมีหน่วยเป็นนาที

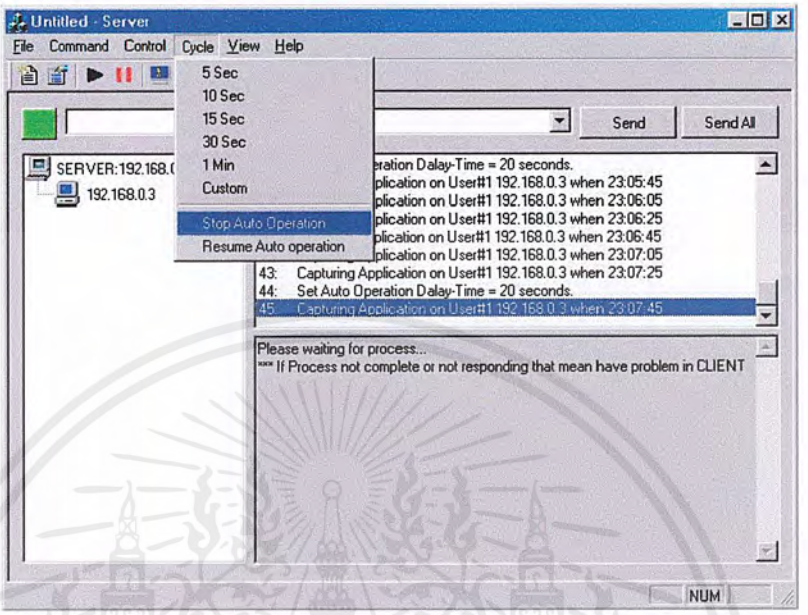
หมายเหตุ โดยปรกติแล้วโปรแกรมเครื่องควบคุมจะทำการกำหนดระยะเวลาการหน่วง (Set Default) ไว้ที่ 20 วินาที



รูปที่ 7.12 แสดงการตรวจจับแอพพลิเคชันแบบ Auto Operation ที่ระยะเวลาการหน่วง 20 วินาที

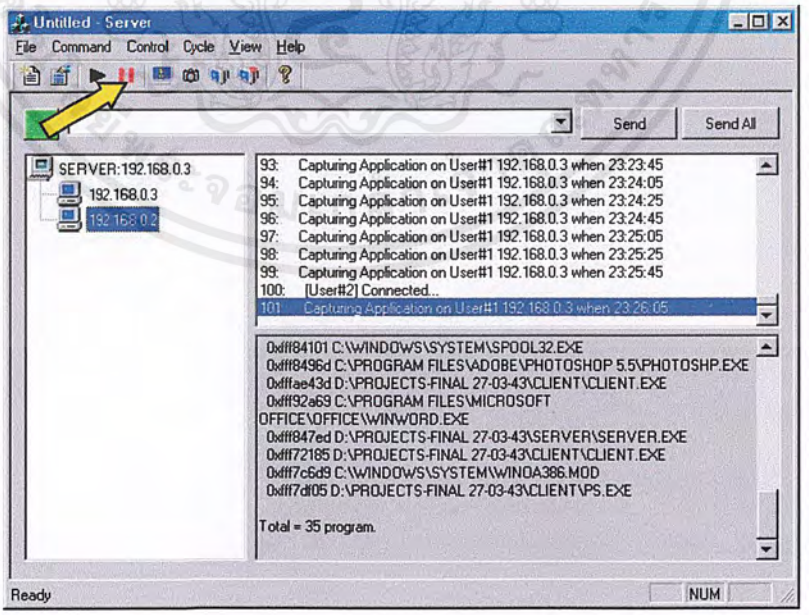
ในกรณีที่ต้องการหยุดการตรวจจับแอพพลิเคชันแบบ Auto Operation เพื่อทำ Manual Operation ต่าง ๆ ต้องทำการขัดจังหวะ (Interrupt) โดยมีวิธีให้เลือก 3 วิธีคือ

วิธีที่ 1 เลือก Stop Auto Operation จากเมนูบาร์



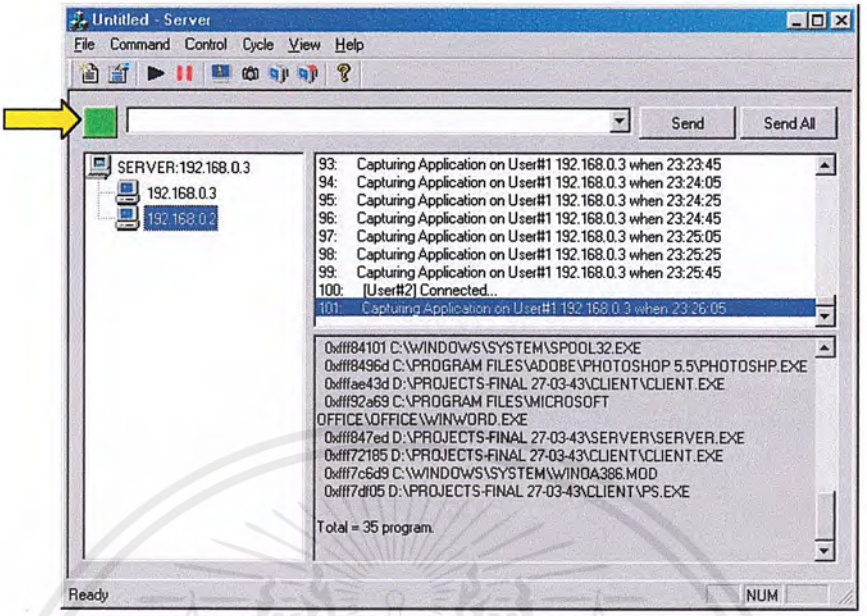
รูปที่ 7.13 แสดงการขัดจังหวะ Auto Operation โดยเลือก Stop Auto Operation

วิธีที่ 2 เลือกจาก Toolbar

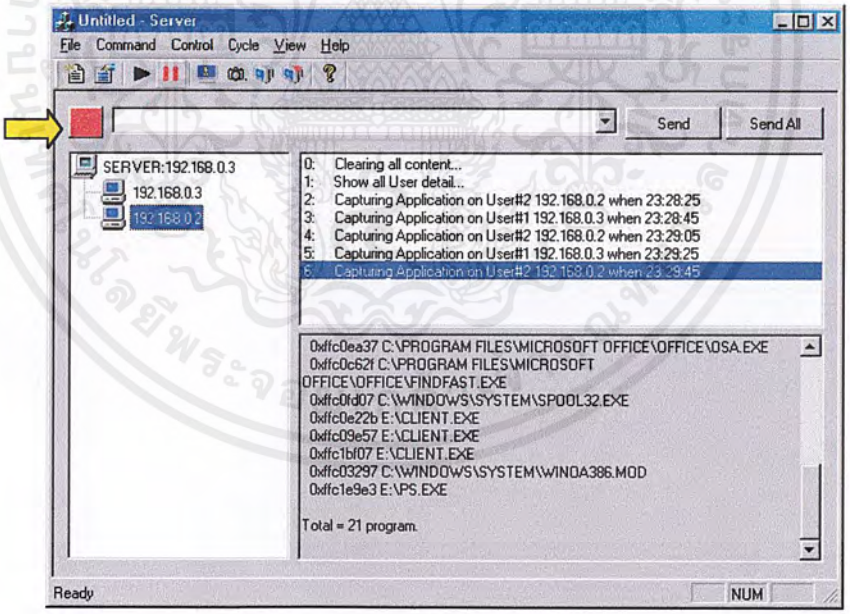


รูปที่ 7.14 แสดงการขัดจังหวะ Auto Operation โดยเลือกจาก Toolbar

วิธีที่ 3 กดปุ่ม Interrupt Process (สีเขียว) เมื่อกดแล้วปุ่มนั้นจะเปลี่ยนเป็นสีแดง



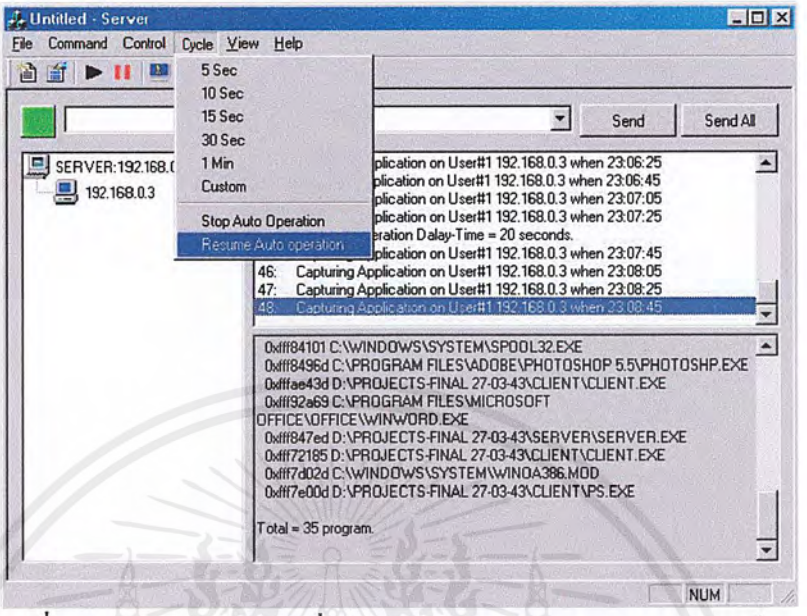
รูปที่ 7.15 แสดงปุ่ม Interrupt Process



รูปที่ 7.16 แสดงปุ่ม Interrupt Process เป็นสีแดง

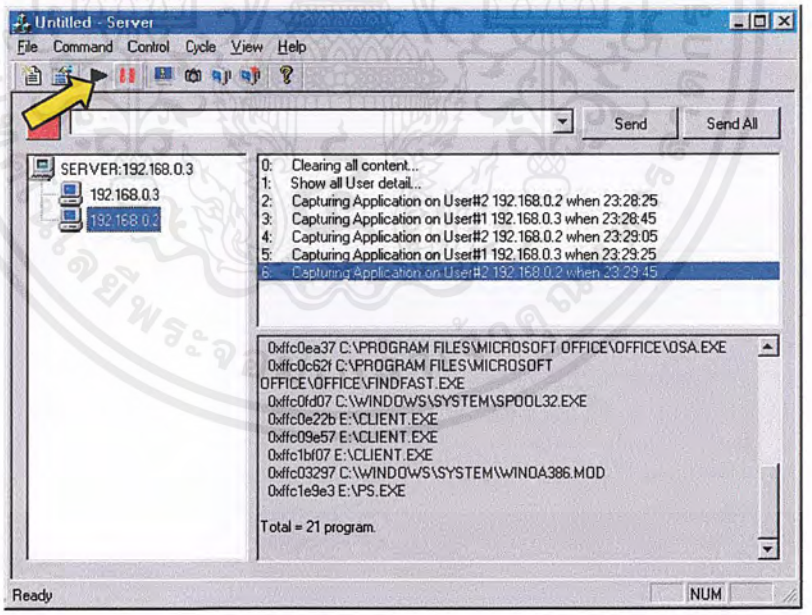
เมื่อต้องการกลับไปตรวจจับแอพพลิเคชันแบบ Auto Operation สามารถทำได้ 2 วิธี คือ

วิธีที่ 1 เลือกคำสั่ง Resume Auto Operation จากเมนูบาร์



รูปที่ 7.17 แสดงการเลือกคำสั่ง Resume Auto Operation จากเมนูบาร์

วิธีที่ 2 เลือกจาก Toolbar



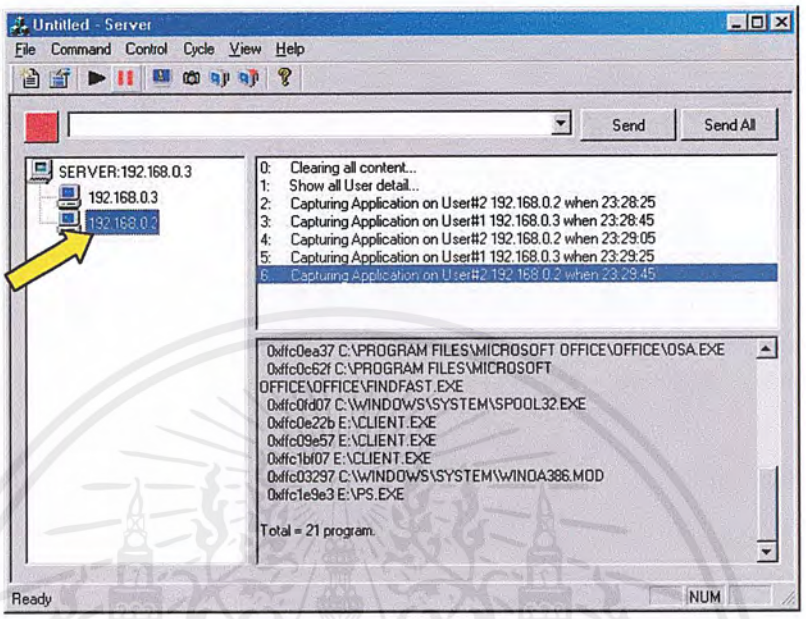
รูปที่ 7.18 แสดงการกดปุ่มที่ Toolbar เพื่อกลับไปตรวจจับแอพพลิเคชันแบบ Auto Operation

หมายเหตุ โดยปรกติแล้วเมื่อทำ Manual Operation เสร็จแล้ว เครื่องจะทำ Auto Operation โดยอัตโนมัติ

แบบที่ 2 การตรวจจับแอพพลิเคชันโดยปรกติ Manual Operation สามารถทำได้มี 2 วิธีคือ

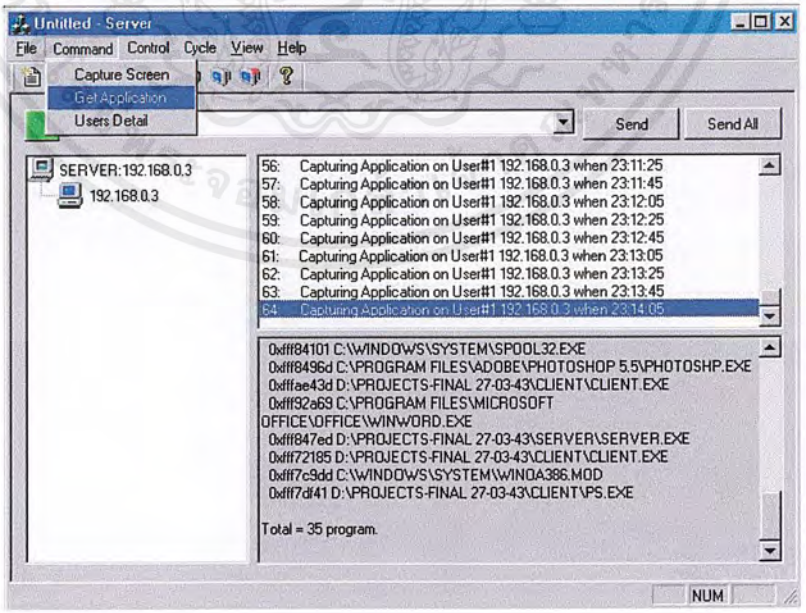
วิธีที่ 1 การเลือกจากเมนูบาร์ มีขั้นตอนดังต่อไปนี้

ขั้นตอนที่ 1 คลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ



รูปที่ 7.19 แสดงการคลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ

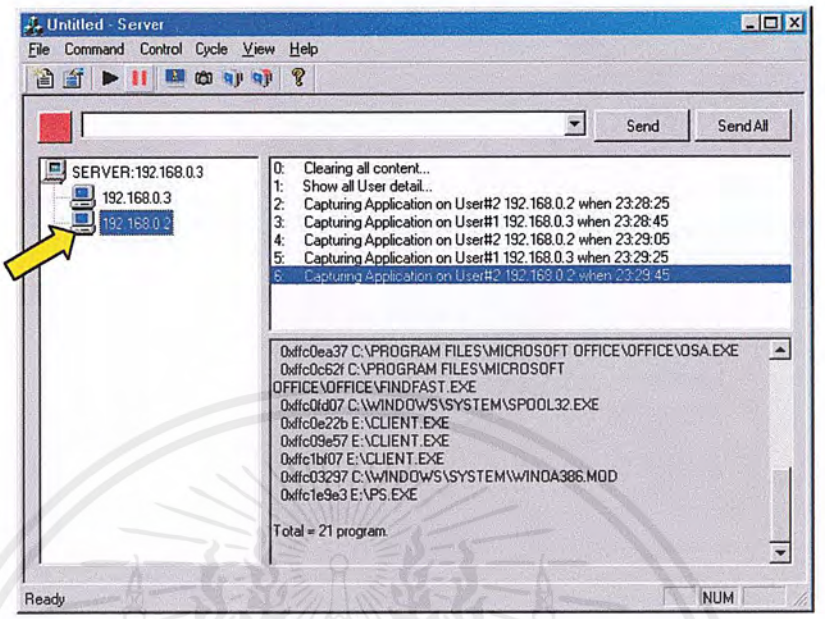
ขั้นตอนที่ 2 เลือกคำสั่ง Get Application ที่เมนูบาร์



รูปที่ 7.20 แสดงการเลือกคำสั่ง Get Application

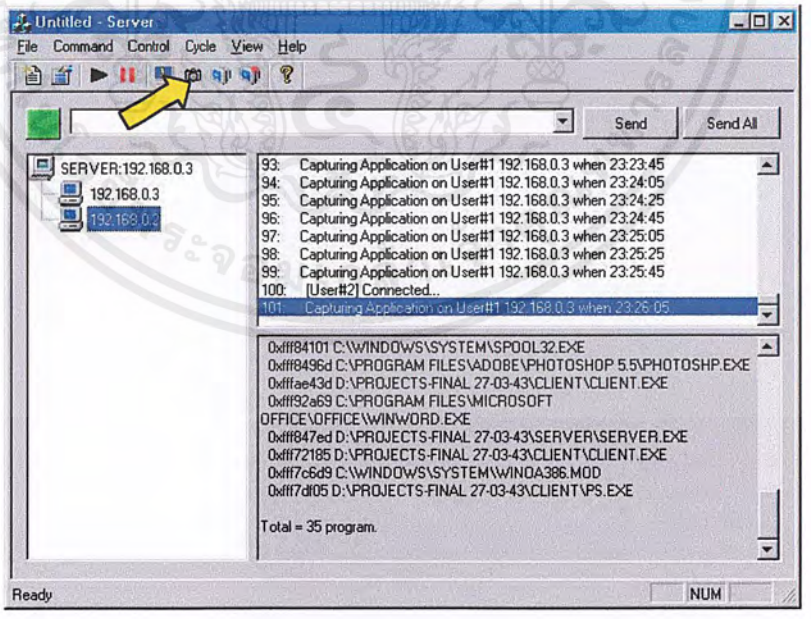
วิธีที่ 2 เลือกจาก Toolbar

ขั้นตอนที่ 1 คลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ



รูปที่ 7.21 แสดงการคลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ

ขั้นตอนที่ 2 เลือกปุ่มที่ Toolbar

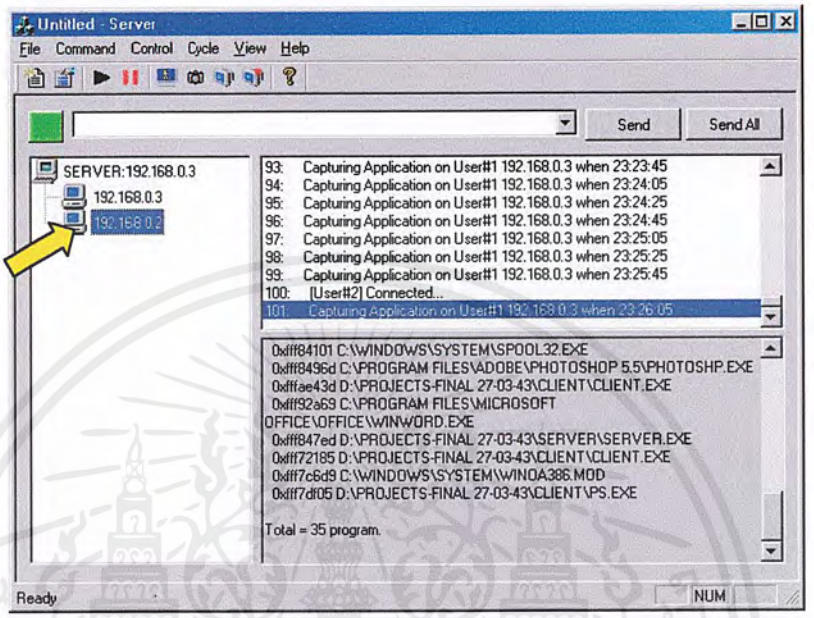


รูปที่ 7.22 แสดงการเลือกปุ่ม Toolbar เพื่อตรวจจับแอพพลิเคชัน

7.2.5 เครื่องควบคุมสั่งให้ทำการจับภาพหน้าจอของเครื่องลูกข่ายมาแสดงที่เครื่องควบคุม สามารถทำได้ 3 วิธีคือ

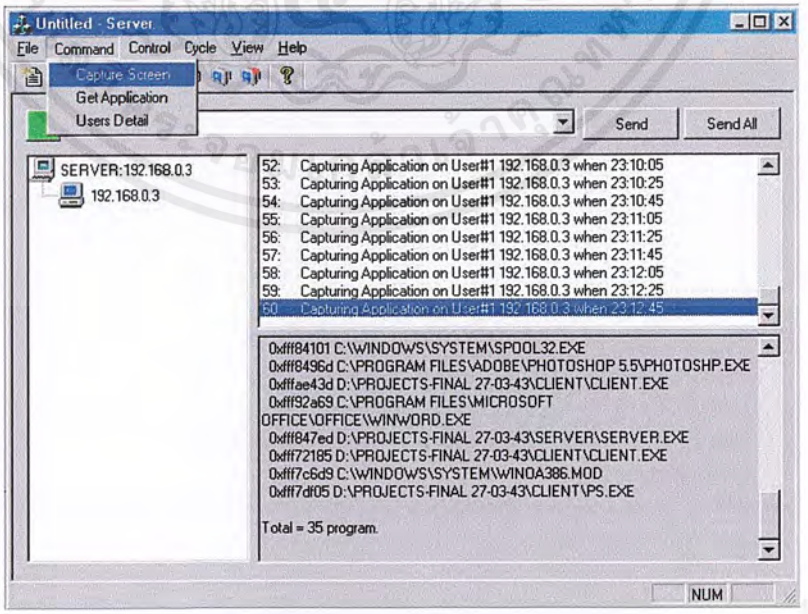
วิธีที่ 1 เลือกคำสั่ง Capture Screen จากเมนูบาร์ มีขั้นตอนดังนี้

ขั้นตอนที่ 1 คลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ



รูปที่ 7.23 แสดงการคลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการตรวจจับหน้าจอ

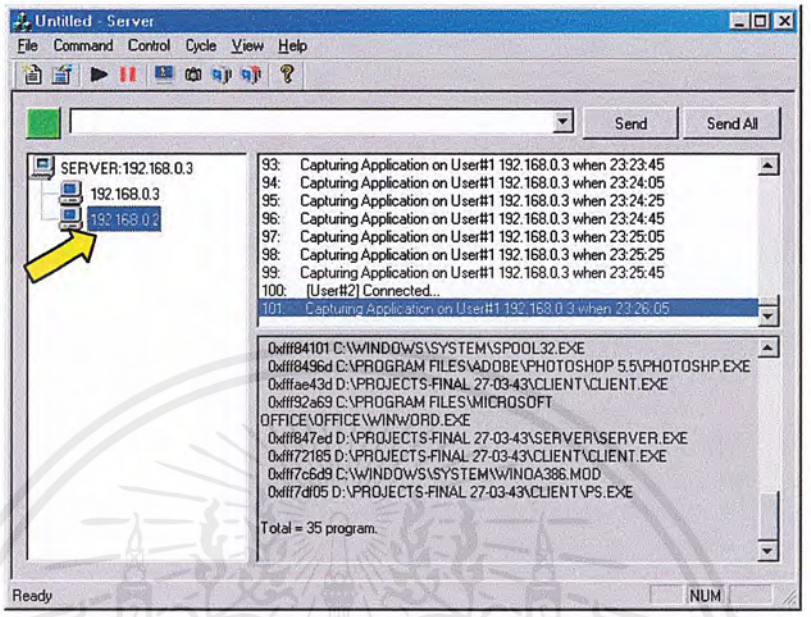
ขั้นตอนที่ 2 เลือกคำสั่ง Capture Screen ที่เมนูบาร์



รูปที่ 7.24 แสดงการเลือกคำสั่ง Capture Screen จากเมนูบาร์

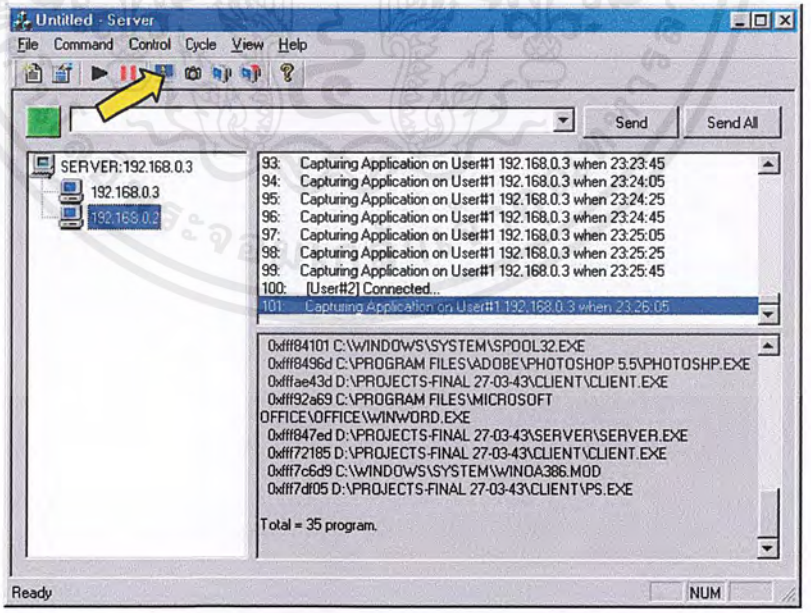
วิธีที่ 2 เลือกจาก Toolbar มีขั้นตอนดังต่อไปนี้

ขั้นตอนที่ 1 คลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ



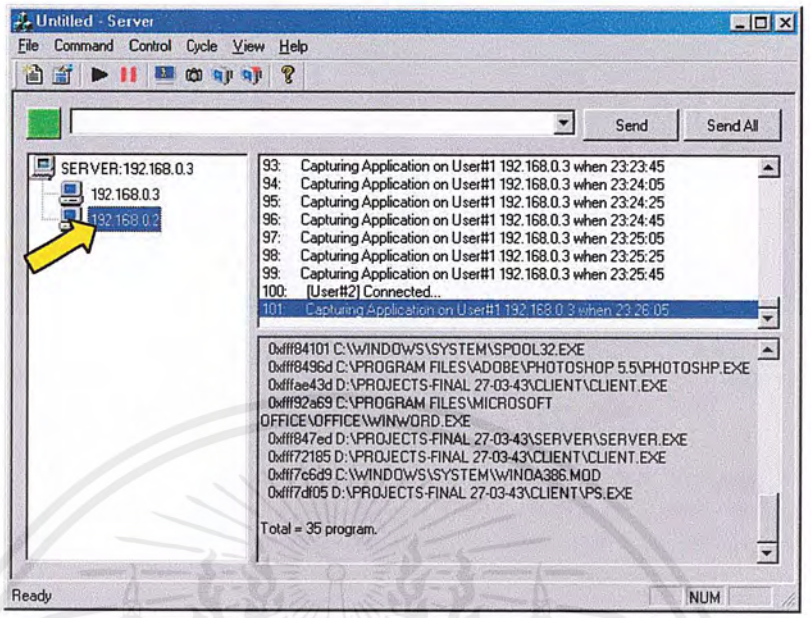
รูปที่ 7.25 แสดงการเลือก IP ของเครื่องลูกข่ายที่ต้องการ

ขั้นตอนที่ 2 เลือกปุ่ม Toolbar เพื่อจับภาพหน้าจอเครื่องลูกข่าย

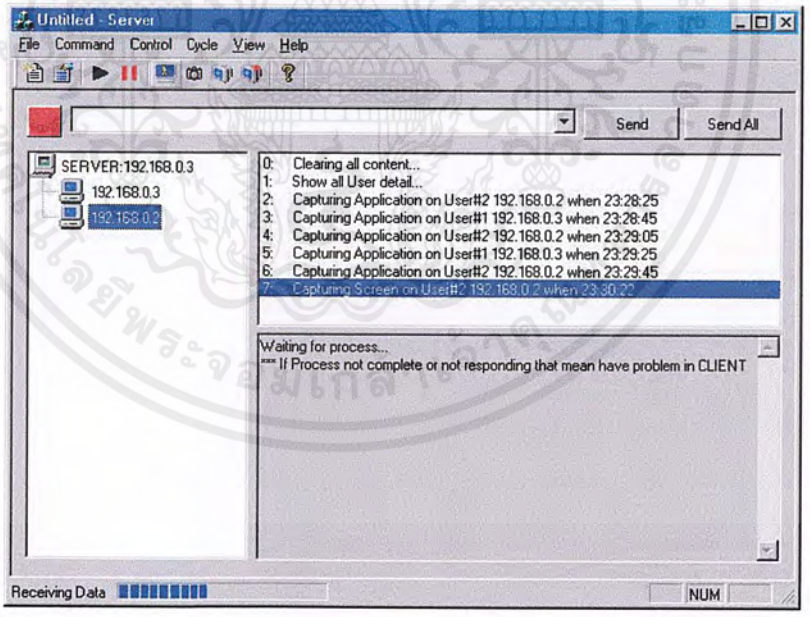


รูปที่ 7.26 แสดงการเลือกปุ่มที่ Toolbar เพื่อจับภาพหน้าจอเครื่องลูกข่าย

วิธีที่ 3 ดับเบิลคลิกที่ IP เครื่องลูกข่ายที่ต้องการ

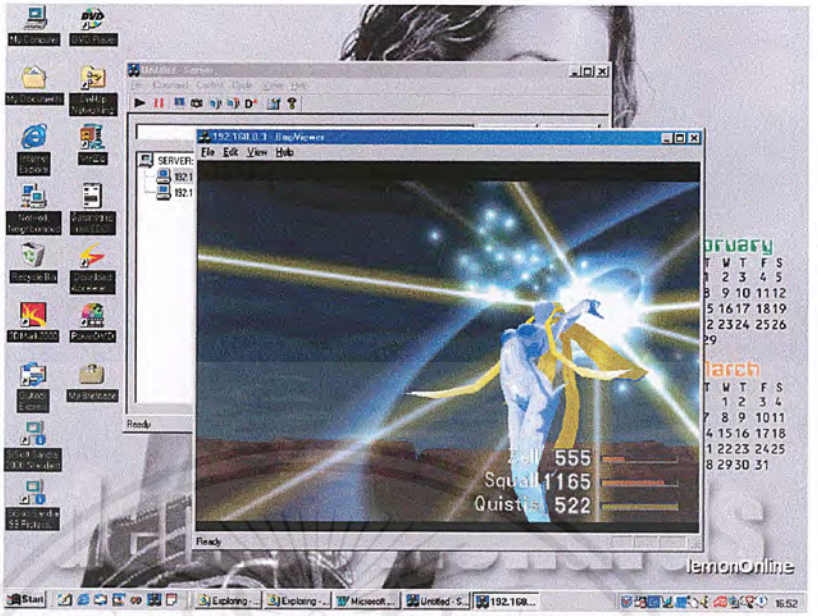


รูปที่ 7.27 แสดงการดับเบิลคลิกที่ IP ของเครื่องลูกข่ายที่ต้องการ



รูปที่ 7.28 แสดงการทำงานของโปรแกรมเครื่องควบคุมขณะที่กำลังจับภาพหน้าจอของเครื่องลูกข่ายที่ต้องการ

เมื่อทำการจับภาพหน้าจอเครื่องลูกข่ายเสร็จแล้วจะมาแสดงที่หน้าจอของเครื่องควบคุม

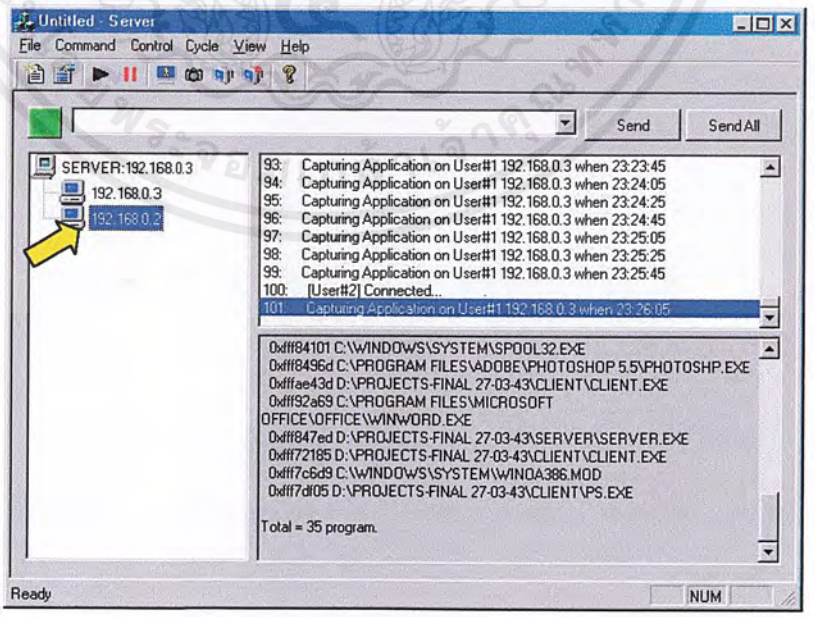


รูปที่ 7.29 แสดงภาพหน้าจอของเครื่องลูกข่ายที่เครื่องควบคุม

7.2.6 เครื่องควบคุมต้องการส่งข้อความไปยังเครื่องลูกข่าย ซึ่งสามารถเลือกที่จะส่งไปยังเครื่องลูกข่ายเพียงเครื่องเดียวที่ต้องการ หรือส่งไปยังเครื่องลูกข่ายทุกเครื่องในห้องปฏิบัติการก็ได้

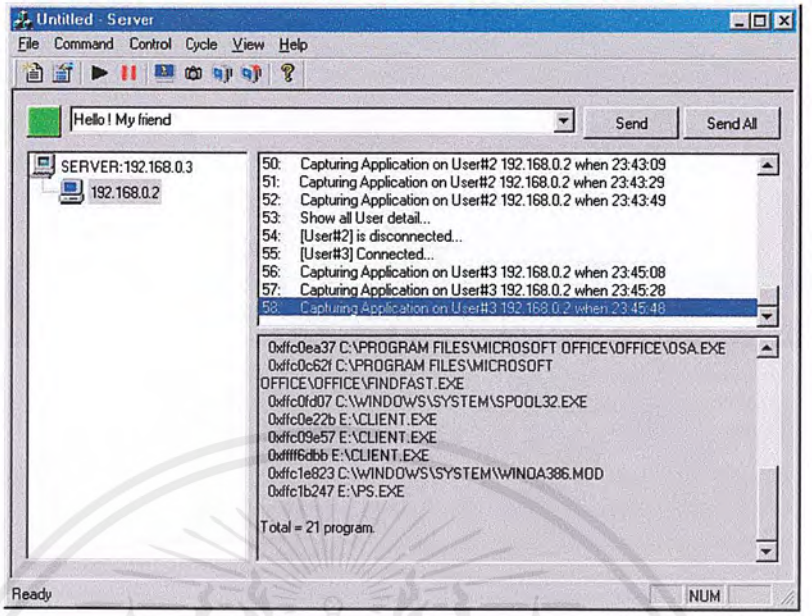
แบบที่ 1 ส่งข้อความไปยังเครื่องลูกข่ายที่ต้องการ มีขั้นตอนดังนี้

ขั้นตอนที่ 1 คลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ



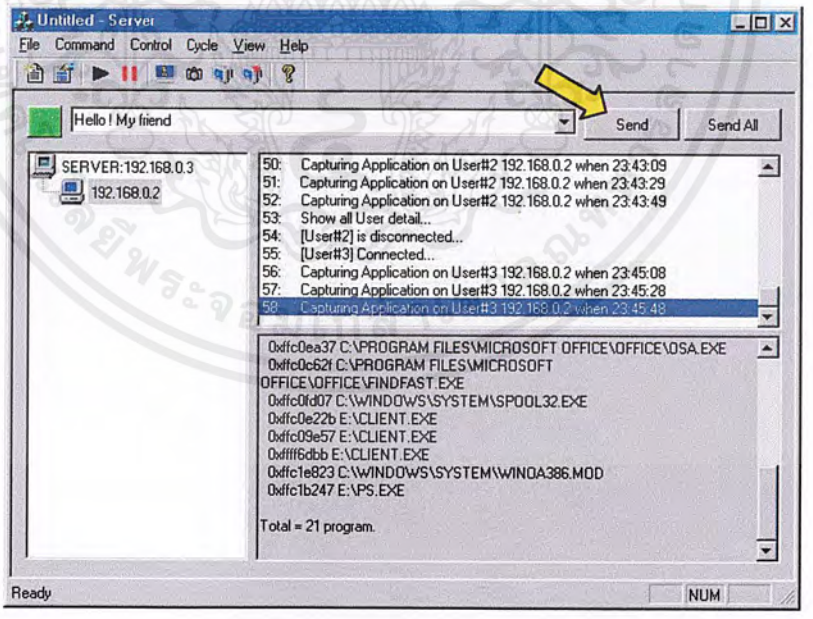
รูปที่ 7.30 คลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ

ขั้นตอนที่ 2 พิมพ์ข้อความที่ต้องการส่งไปยังเครื่องลูกข่ายลงใน Combo Box



รูปที่ 7.31 แสดงการพิมพ์ข้อความที่ต้องการส่งลงใน Combo Box

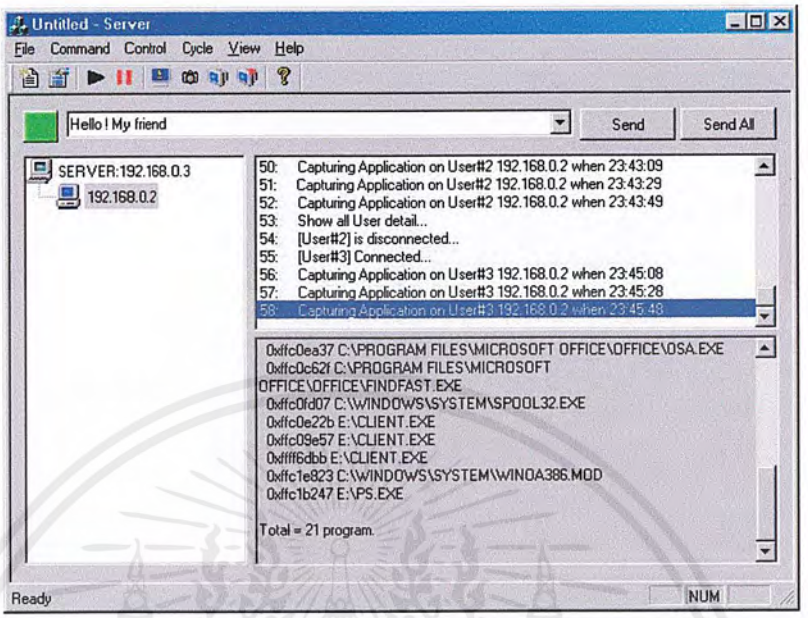
ขั้นตอนที่ 3 คลิกปุ่ม Send เพื่อส่งข้อความนั้นไปยังเครื่องลูกข่าย



รูปที่ 7.32 แสดงการคลิกที่ปุ่ม Send เพื่อส่งข้อความไปยังเครื่องลูกข่าย

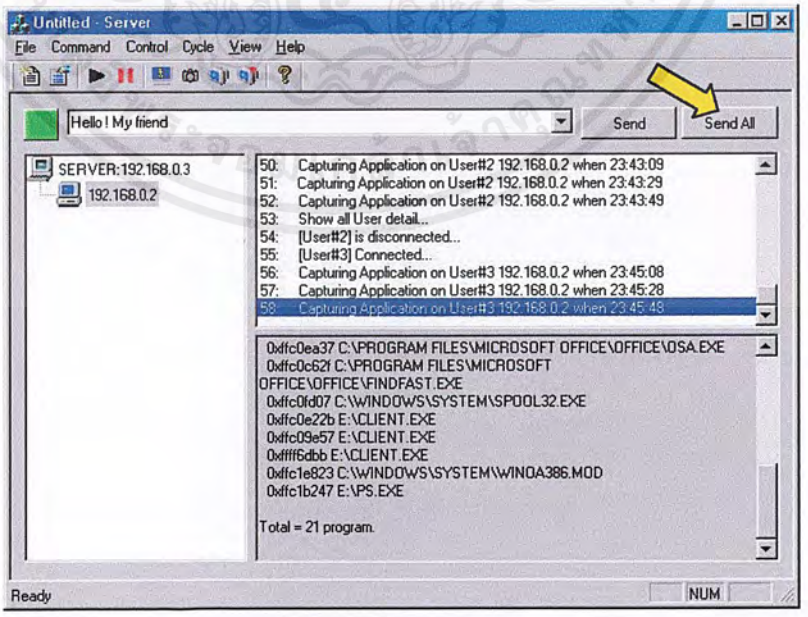
แบบที่ 2 ส่งข้อความไปยังเครื่องลูกข่ายทุกเครื่องในห้องปฏิบัติการ มีขั้นตอนดังนี้

ขั้นตอนที่ 1 พิมพ์ข้อความที่ต้องการส่งไปยังเครื่องลูกข่ายลงใน Combo Box



รูปที่ 7.33 แสดงการพิมพ์ข้อความที่ต้องการส่งไปยังเครื่องลูกข่ายลงใน Combo Box

ขั้นตอนที่ 2 คลิกปุ่ม Send All เพื่อส่งข้อความนั้นไปยังเครื่องลูกข่ายทุกเครื่องในห้องปฏิบัติการ



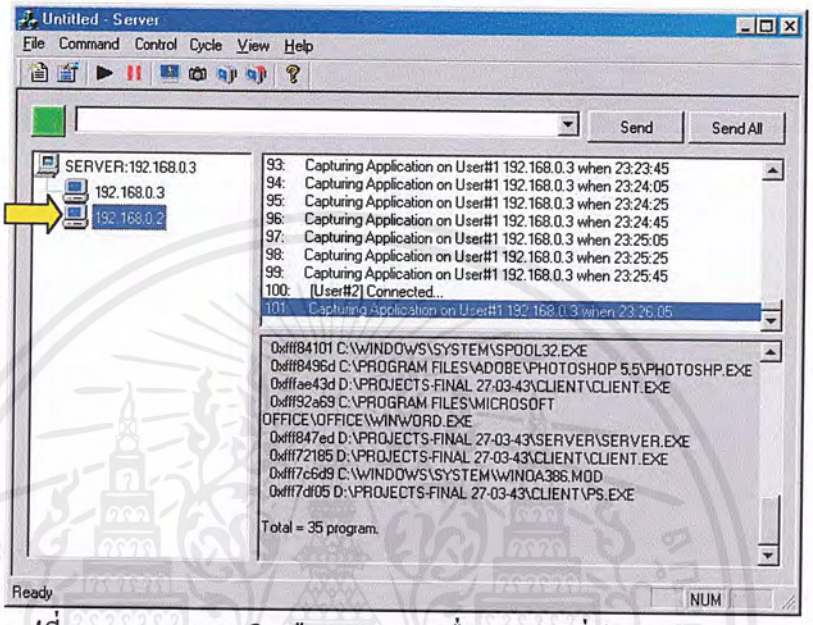
รูปที่ 7.34 แสดงการคลิกปุ่ม Send All เพื่อส่งข้อความ

7.2.7 เครื่องควบคุมต้องการปิดระบบปฏิบัติการของเครื่องลูกข่ายในกรณีที่เครื่องลูกข่ายทำผิดกฎ ซึ่งสามารถเลือกที่จะปิดเครื่องลูกข่ายเพียงเครื่องเดียวหรือปิดทุกเครื่องก็ได้

แบบที่ 1 ต้องการปิดระบบปฏิบัติการของเครื่องลูกข่ายเพียงเครื่องเดียว มี 2 วิธีคือ

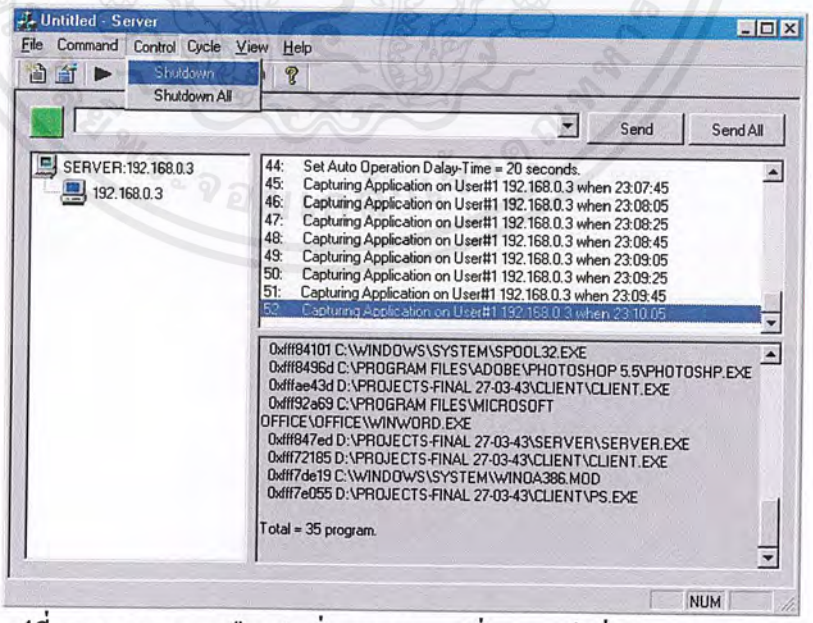
วิธีที่ 1 เลือกคำสั่ง Shutdown จากเมนูบาร์ มีขั้นตอนดังนี้

ขั้นตอนที่ 1 คลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ



รูปที่ 7.35 แสดงการคลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ

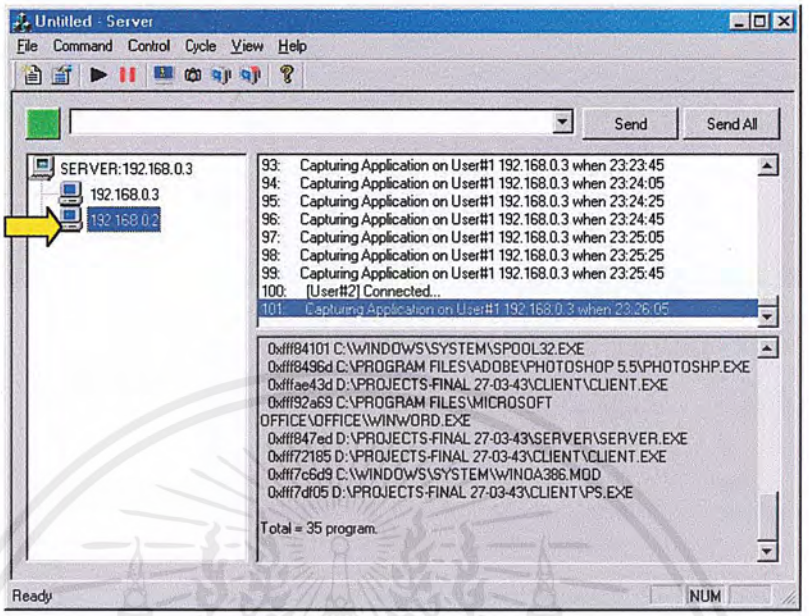
ขั้นตอนที่ 2 เลือกคำสั่ง Shutdown ที่เมนูบาร์



รูปที่ 7.36 แสดงการเลือกคำสั่ง Shutdown ที่เมนูบาร์ เพื่อปิดระบบปฏิบัติการของเครื่องลูกข่ายนั้น

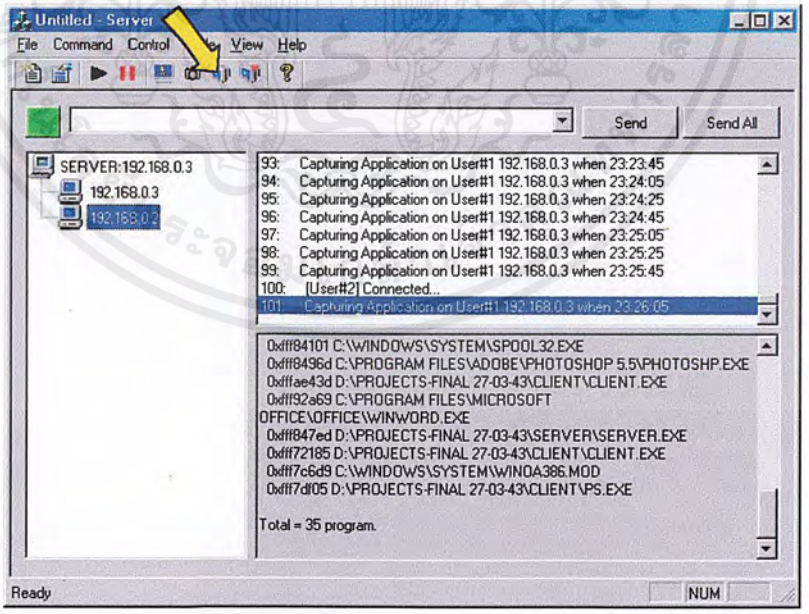
วิธีที่ 2 เลือกจาก Toolbar มีขั้นตอนดังนี้

ขั้นตอนที่ 1 คลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ



รูปที่ 7.37 แสดงการคลิกเลือก IP ของเครื่องลูกข่ายที่ต้องการ

ขั้นตอนที่ 2 เลือกปุ่มที่ Toolbar

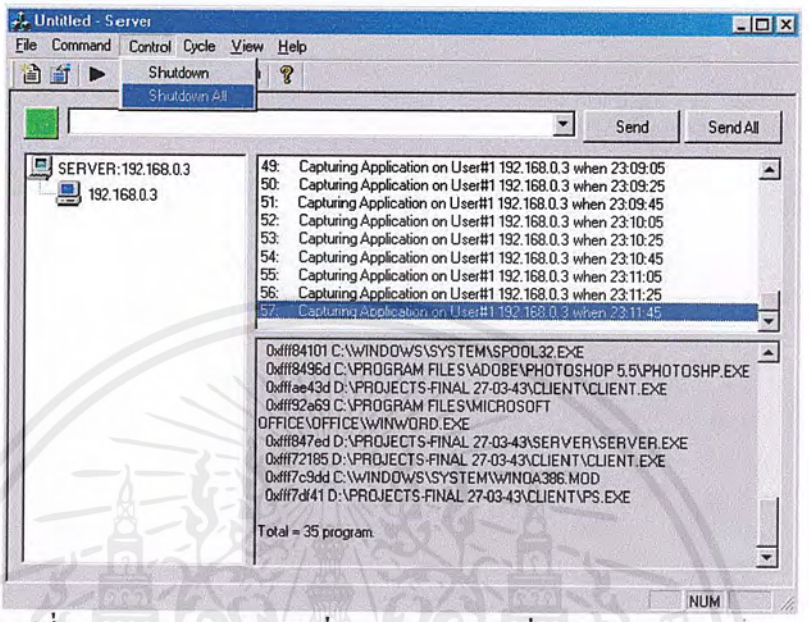


รูปที่ 7.38 แสดงการเลือกปุ่มที่ Toolbar เพื่อปิดระบบปฏิบัติการของเครื่องลูกข่ายที่ต้องการ

แบบที่ 2 สั่งปิดระบบปฏิบัติการของเครื่องลูกข่ายทุกเครื่องในห้องปฏิบัติการ มี2 วิธี คือ

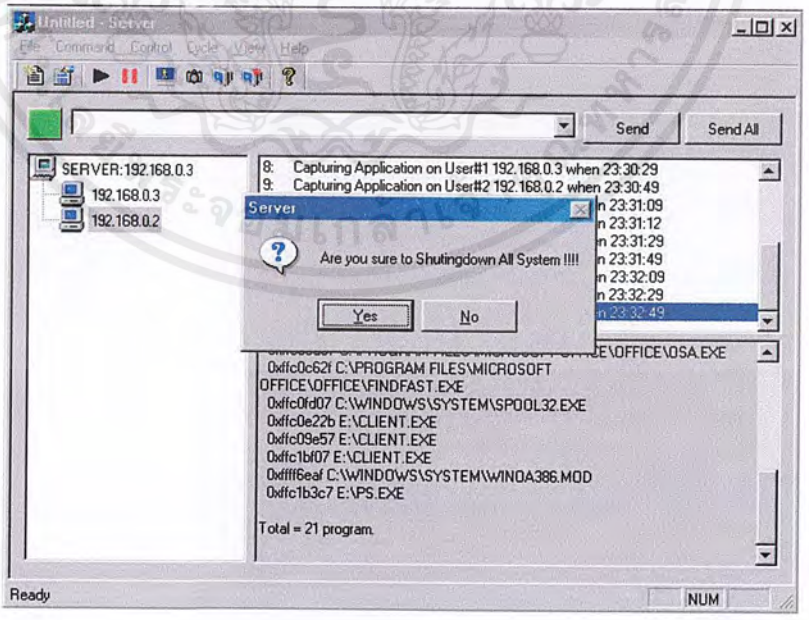
วิธีที่ 1 เลือกคำสั่ง Shutdown All จากเมนูบาร์ มีขั้นตอนดังนี้

ขั้นตอนที่ 1 เลือกคำสั่ง Shutdown All จากเมนูบาร์



รูปที่ 7.39 แสดงการเลือกคำสั่ง Shutdown All ที่เมนูบาร์

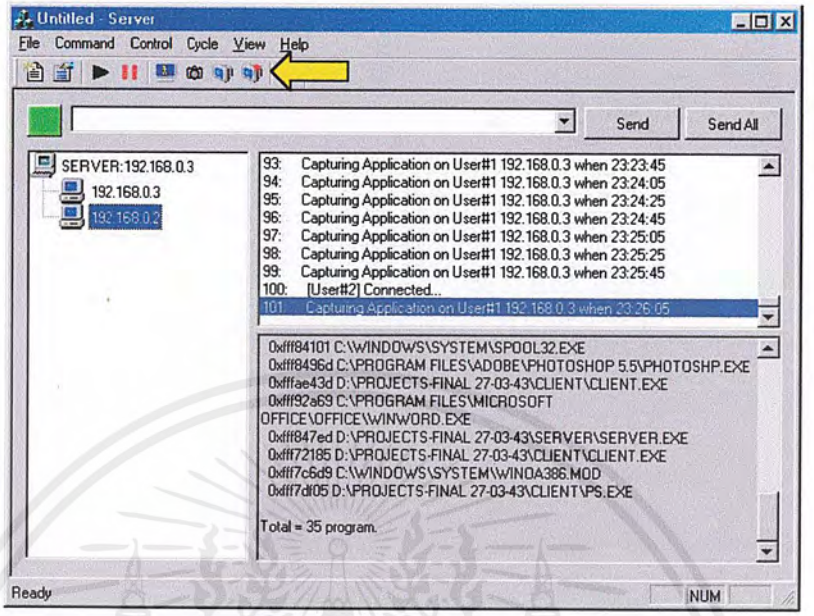
ขั้นตอนที่ 2 เมื่อเลือกคำสั่งดังกล่าวจะปรากฏข้อความเพื่อยืนยันคำสั่ง แล้วคลิก OK



รูปที่ 7.40 แสดงข้อความยืนยันคำสั่งให้ปิดระบบปฏิบัติการของเครื่องลูกข่ายทุกเครื่อง

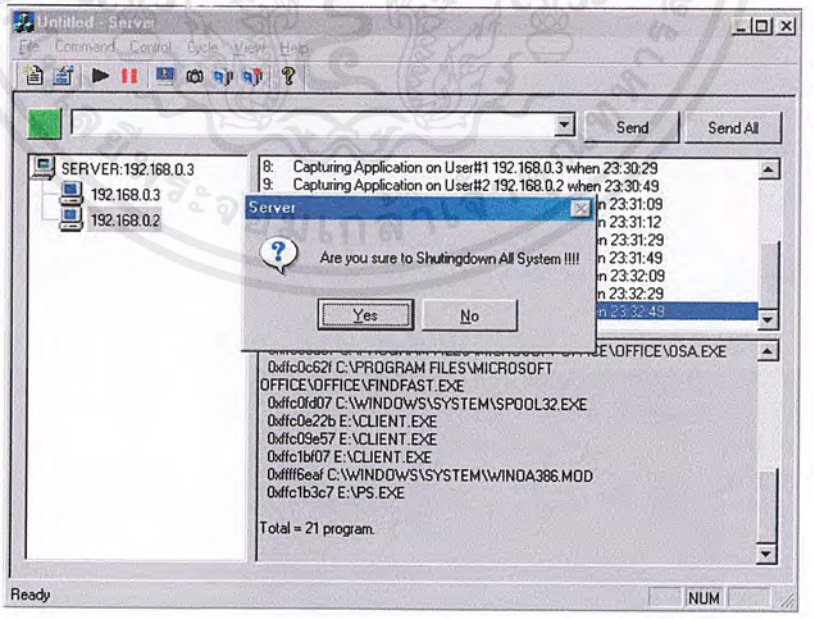
วิธีที่ 2 เลือกปุ่ม Shutdown All จากปุ่มที่ Toolbar มีขั้นตอนดังนี้

ขั้นตอนที่ 1 เลือกปุ่ม Shutdown All จากปุ่มที่ Toolbar



รูปที่ 7.41 แสดงการเลือกปุ่ม Shutdown All ที่ Toolbar

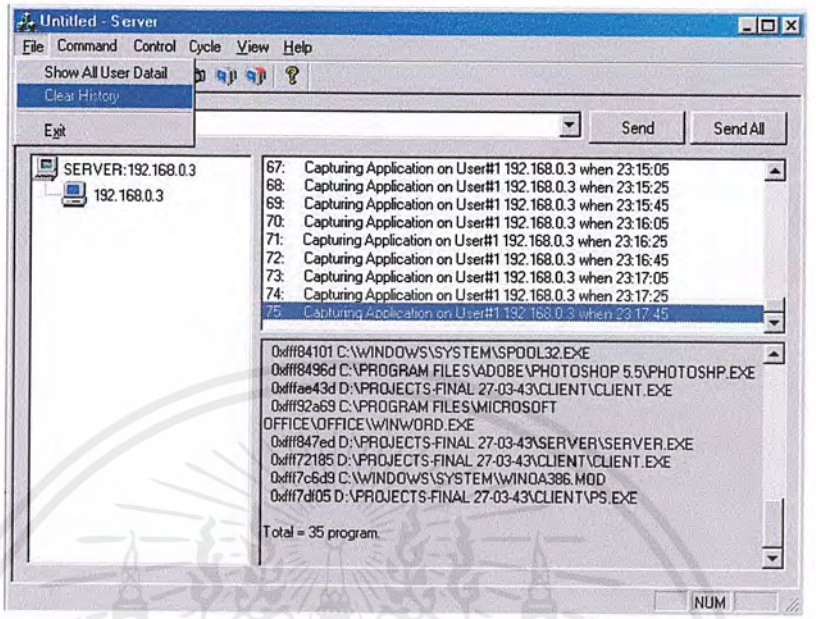
ขั้นตอนที่ 2 เมื่อเลือกคำสั่งดังกล่าวจะเหมือนวิธีที่ 1 คือปรากฏข้อความเพื่อยืนยันคำสั่ง แล้วคลิก OK



รูปที่ 7.42 แสดงข้อความยืนยันคำสั่งให้ปิดระบบปฏิบัติการของเครื่องลูกข่ายทุกเครื่อง

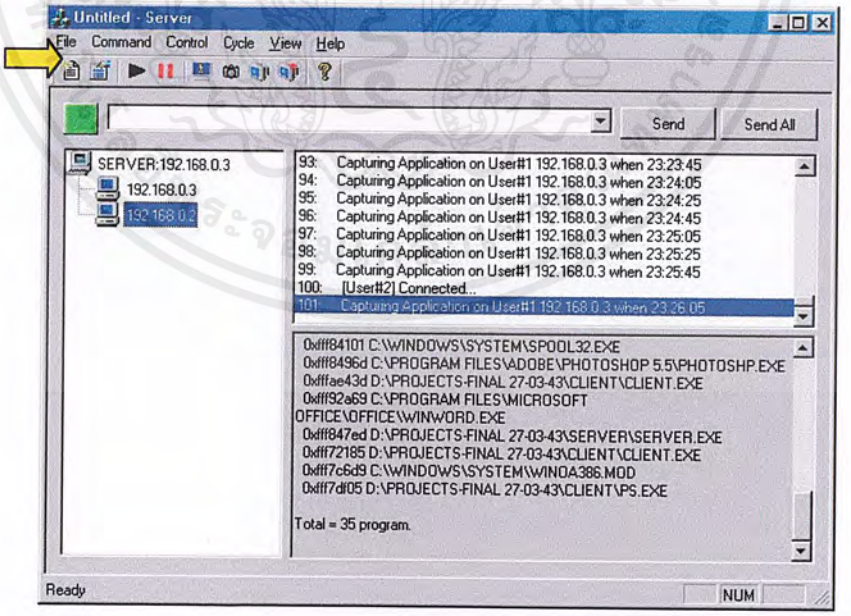
7.2.8 ในกรณีที่เครื่องควบคุมต้องการที่จะ Clear History สามารถกระทำได้ 2 วิธีคือ

วิธีที่ 1 เลือกคำสั่ง Clear History จากเมนูบาร์



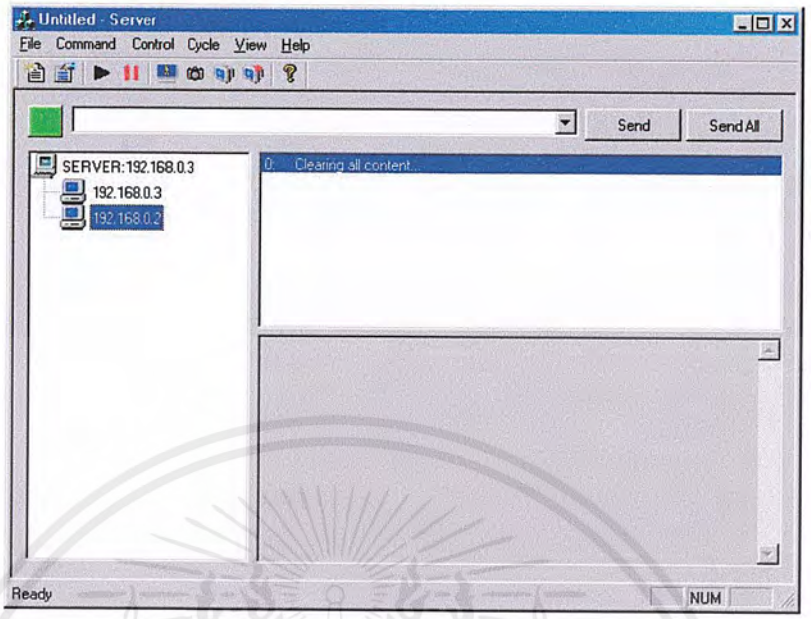
รูปที่ 7.43 แสดงการเลือกคำสั่ง Clear History จากเมนูบาร์

วิธีที่ 2 เลือกปุ่ม Clear History จาก Toolbar



รูปที่ 7.44 แสดงการเลือกปุ่ม Clear History จาก Toolbar

เมื่อทำตามวิธีดังกล่าวโปรแกรมเครื่องควบคุมจะปรากฏผลดังนี้

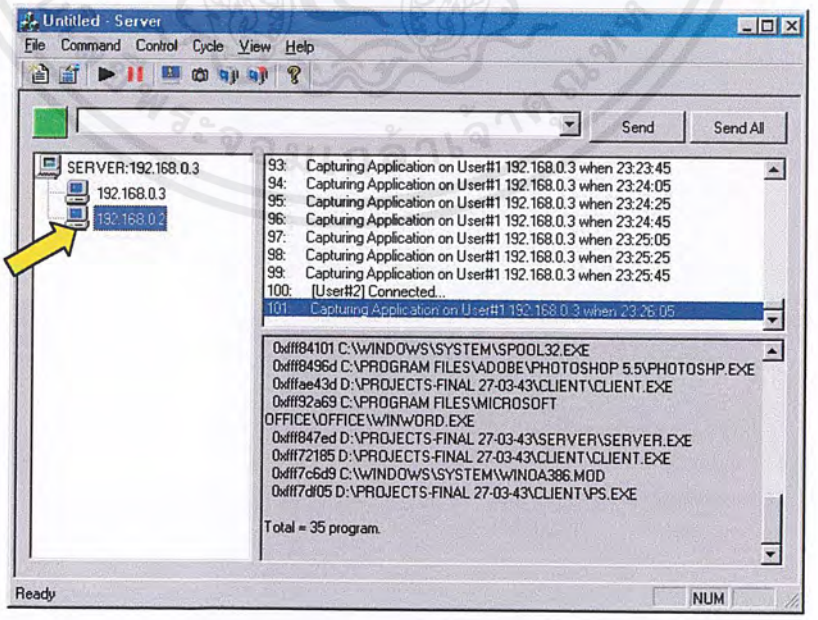


รูปที่ 7.45 แสดงผลที่เกิดจากการ Clear History

7.2.9 เมื่อเครื่องควบคุมต้องการดูรายละเอียด (Information) ของเครื่องลูกข่าย เช่น หมายเลข IP หมายเลข Port ระบบปฏิบัติการ เวลาในการเข้าสู่ระบบ เป็นต้น สามารถทำได้ 2 วิธี คือ

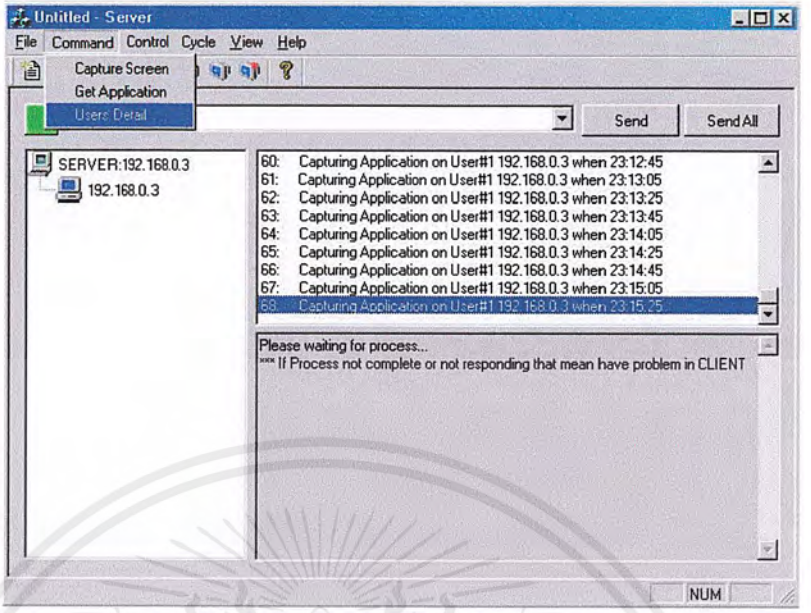
วิธีที่ 1 เลือกคำสั่ง User Detail จากเมนูบาร์ มีขั้นตอนดังนี้

ขั้นตอนที่ 1 เลือก IP ของเครื่องลูกข่ายที่ต้องการดูรายละเอียด



รูปที่ 7.46 แสดงการเลือก IP ของเครื่องลูกข่ายที่ต้องการดูรายละเอียด

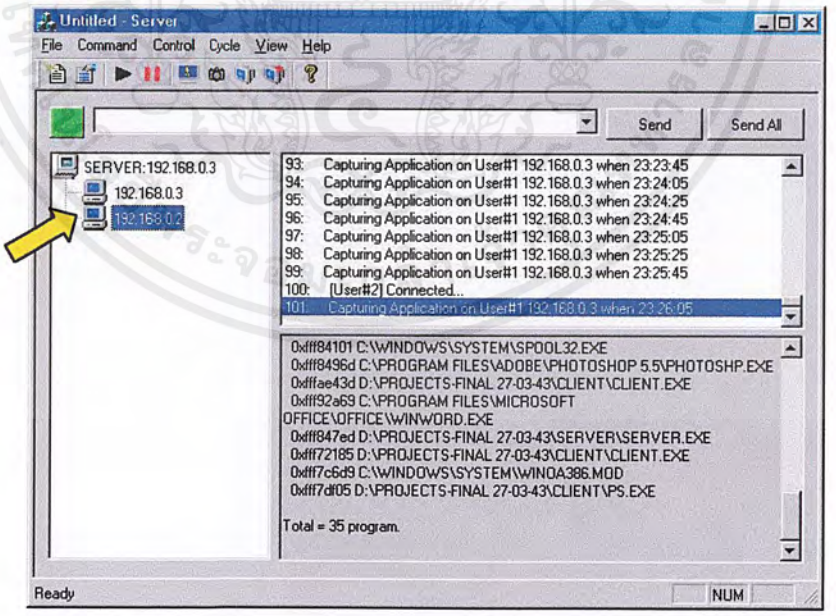
ขั้นตอนที่ 2 เลือกคำสั่ง User Detail ที่เมนูบาร์



รูปที่ 7.47 แสดงการเลือกคำสั่ง User Detail ที่เมนูบาร์

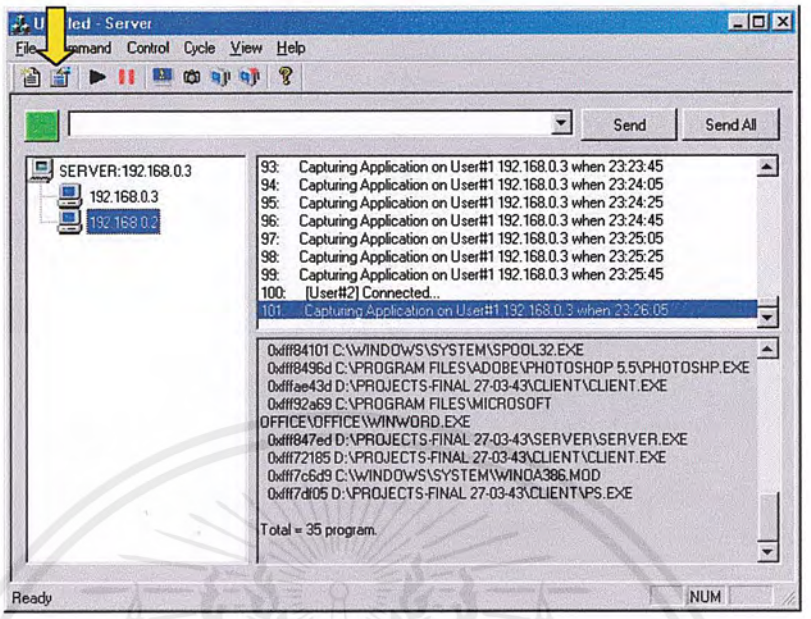
วิธีที่ 2 เลือกปุ่ม User Detail จาก Toolbar มีขั้นตอนดังนี้

ขั้นตอนที่ 1 เลือก IP ของเครื่องลูกข่ายที่ต้องการ



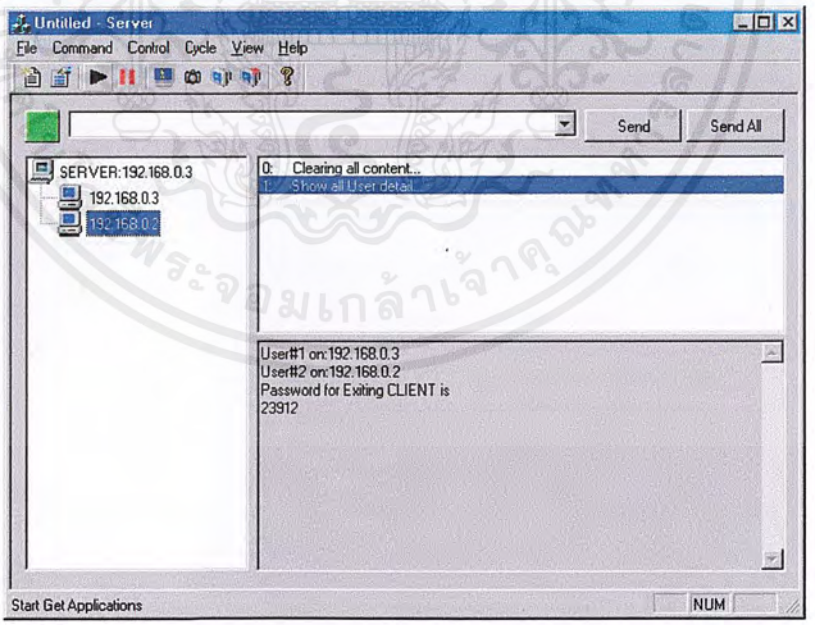
รูปที่ 7.48 แสดงการเลือก IP ของเครื่องที่ต้องการ

ขั้นตอนที่ 2 เลือกปุ่ม User Detail จาก Toolbar



รูปที่ 7.49 แสดงการเลือกปุ่ม User Detail จาก Toolbar

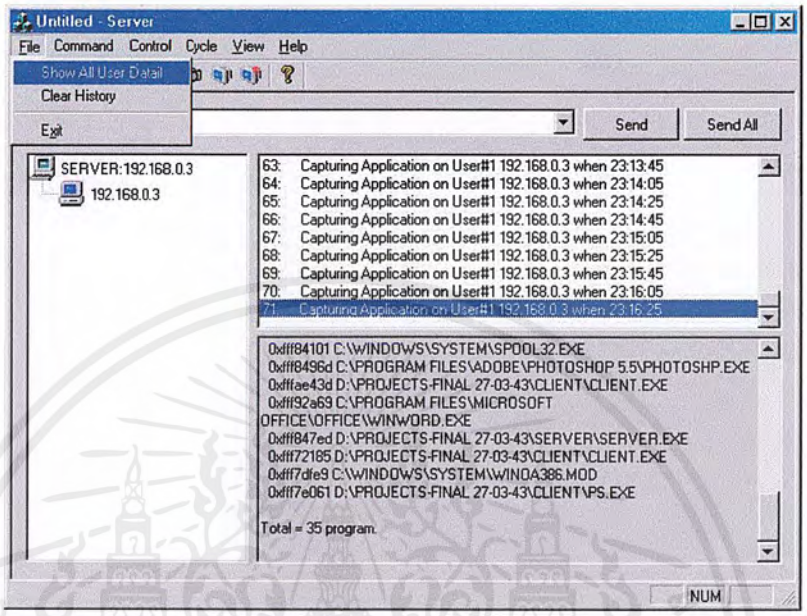
เมื่อทำดังทั้งสองวิธีข้างต้นโปรแกรมเครื่องควบคุมจะมีลักษณะดังนี้



รูปที่ 7.50 แสดงรายละเอียดของ user

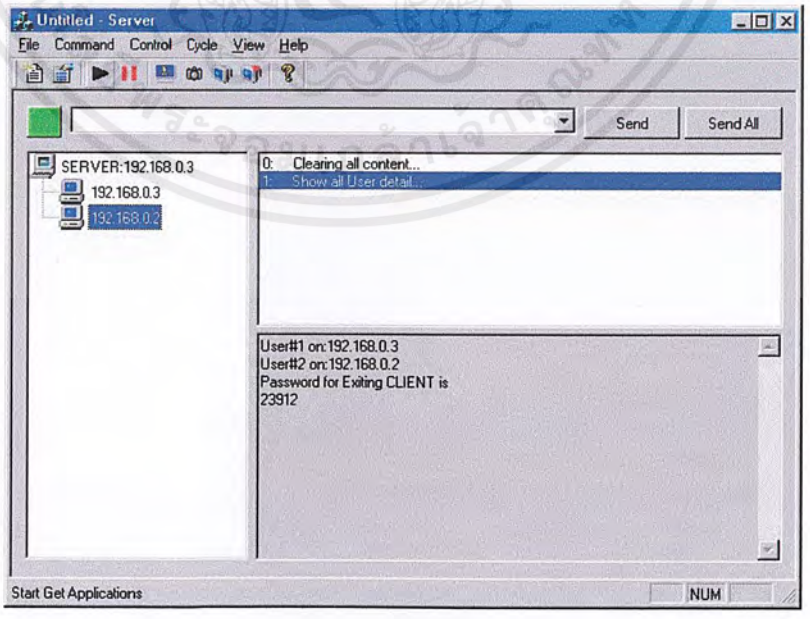
7.2.10 เมื่อต้องการดูว่ามีเครื่องลูกข่ายเครื่องใดบ้างอยู่ในระบบ หมายเลข IP เท่าใด และ Password อะไร มีขั้นตอนดังนี้

ขั้นตอนที่ 1 เลือกคำสั่ง Show All User Detail ที่เมนูบาร์



รูปที่ 7.51 แสดงการเลือกคำสั่ง Show All User Detail ที่เมนูบาร์

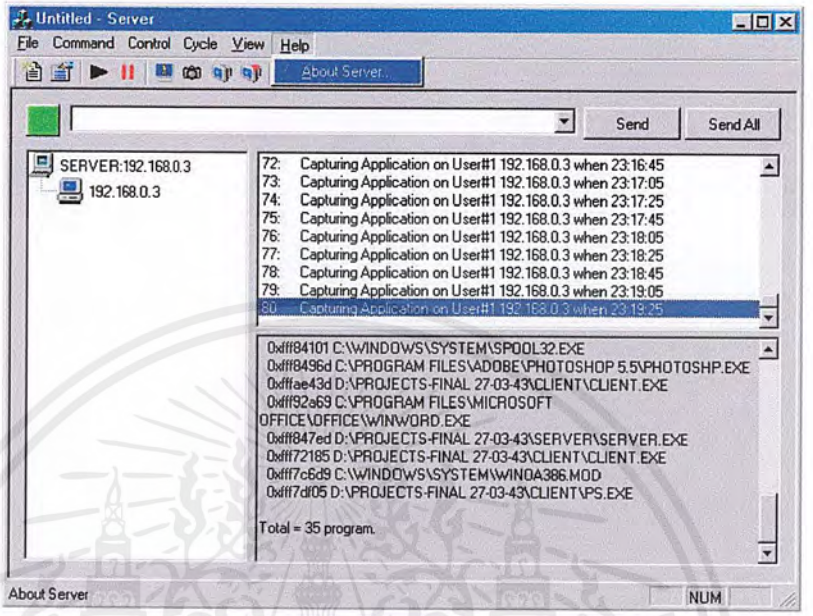
ขั้นตอนที่ 2 เมื่อเลือกคำสั่งดังกล่าว โปรแกรมเครื่องควบคุมจะมีลักษณะดังนี้



รูปที่ 7.52 แสดงโปรแกรมเครื่องควบคุมเมื่อเลือกคำสั่ง Show All User Detail

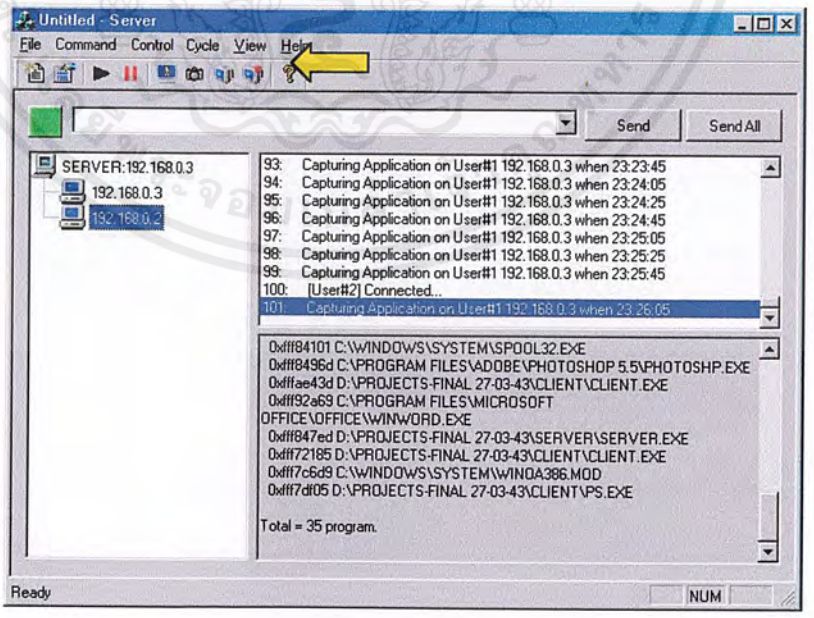
7.2.11 เมื่อต้องการดูเกี่ยวกับรายละเอียดต่าง ๆ ของโปรแกรมเครื่องควบคุม เช่น ชื่อทีมงานผู้จัดทำโปรแกรม เป็นต้น สามารถดูได้ 2 วิธีคือ

วิธีที่ 1 เลือกคำสั่ง About Server จากเมนูบาร์



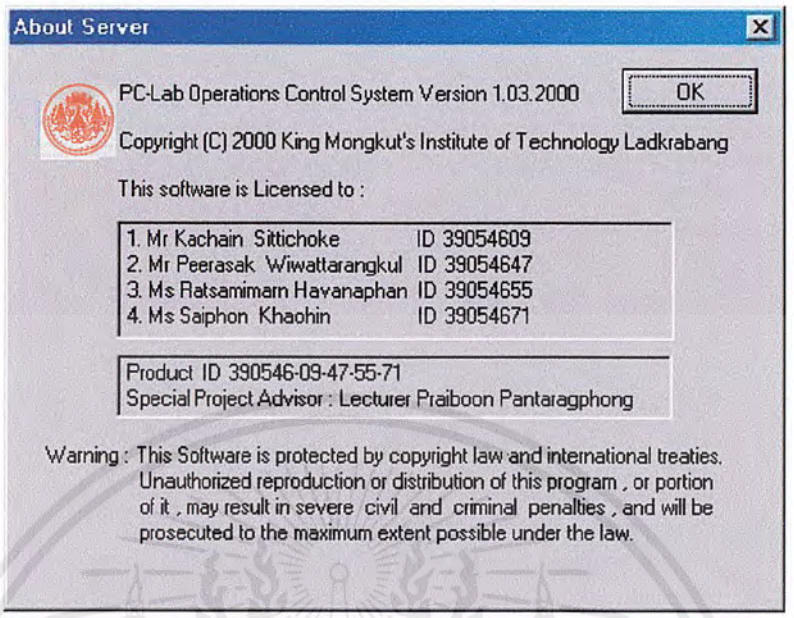
รูปที่ 7.53 แสดงการเลือกคำสั่ง About Server ที่เมนูบาร์

วิธีที่ 2 เลือกปุ่ม About Server จาก Toolbar



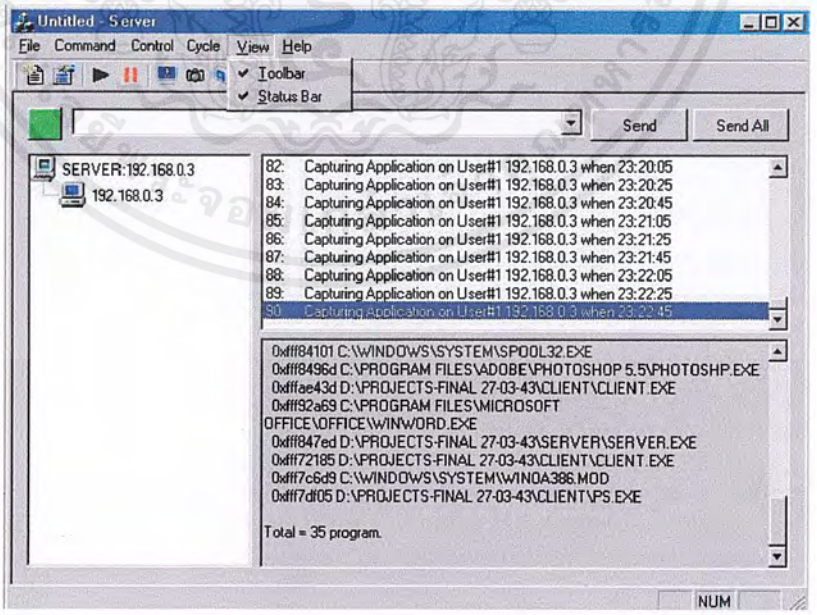
รูปที่ 7.54 แสดงการเลือกปุ่ม About Server จาก Toolbar

เมื่อทำตามวิธีดังกล่าวข้างต้น โปรแกรมเครื่องควบคุมจะปรากฏข้อความดังนี้



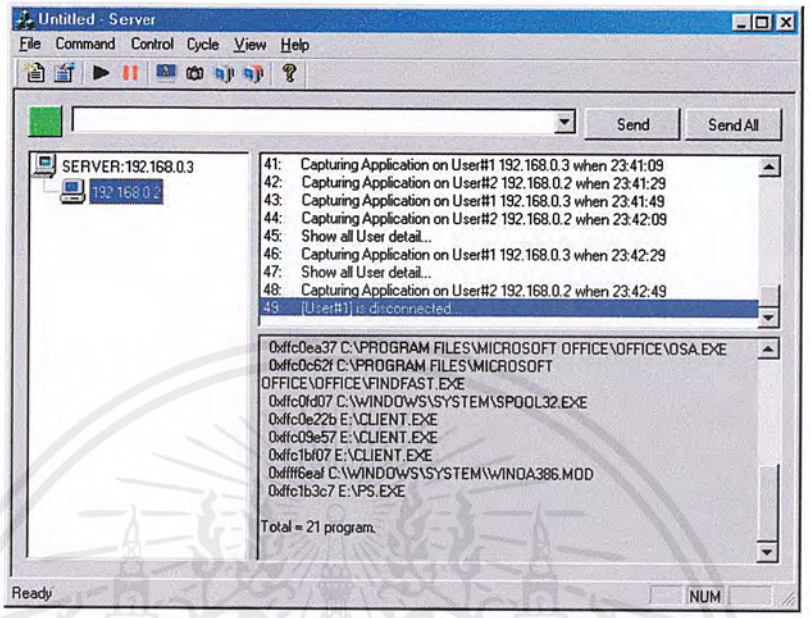
รูปที่ 7.55 แสดงข้อมูลและรายละเอียดเกี่ยวกับโปรแกรมเครื่องควบคุม

7.2.12 เมื่อคลิกเลือกคำสั่ง View ที่เมนูบาร์ จะแสดงแถบเครื่องมือที่โปรแกรมเครื่องควบคุมกำลังใช้งานอยู่

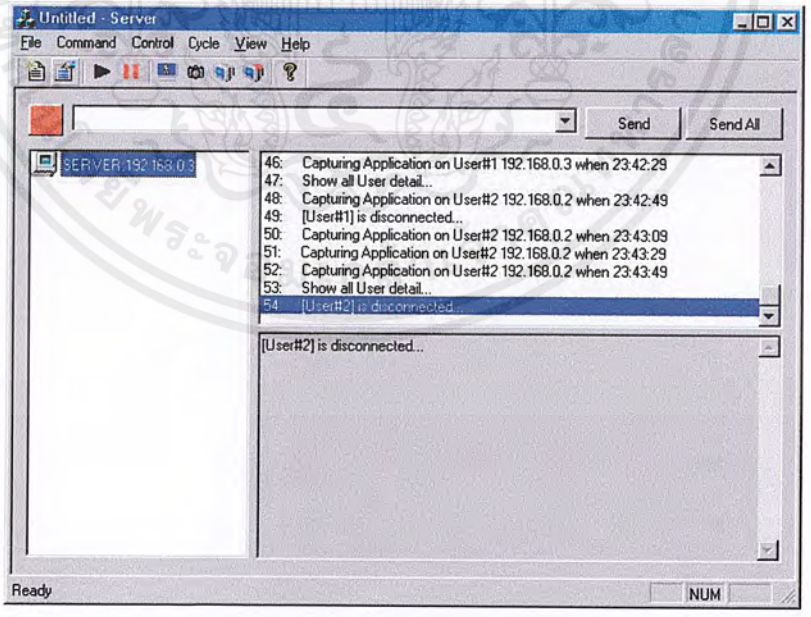


รูปที่ 7.56 แสดงการเลือกคำสั่ง View ที่เมนูบาร์

7.2.13 เมื่อเครื่องลูกข่ายในห้องปฏิบัติการออกจากระบบโปรแกรมเครื่องควบคุมจะปรากฏข้อความดังนี้



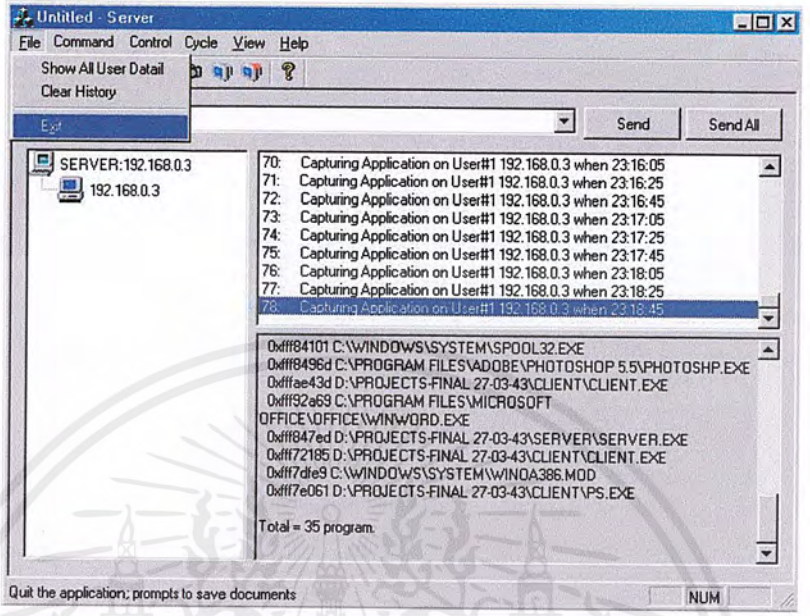
รูปที่ 7.57 แสดงโปรแกรมเครื่องควบคุมเมื่อเครื่องลูกข่ายเครื่องแรกออกจากระบบ



รูปที่ 7.58 แสดงโปรแกรมเครื่องควบคุมเมื่อเครื่องลูกข่ายเครื่องที่สองออกจากระบบ

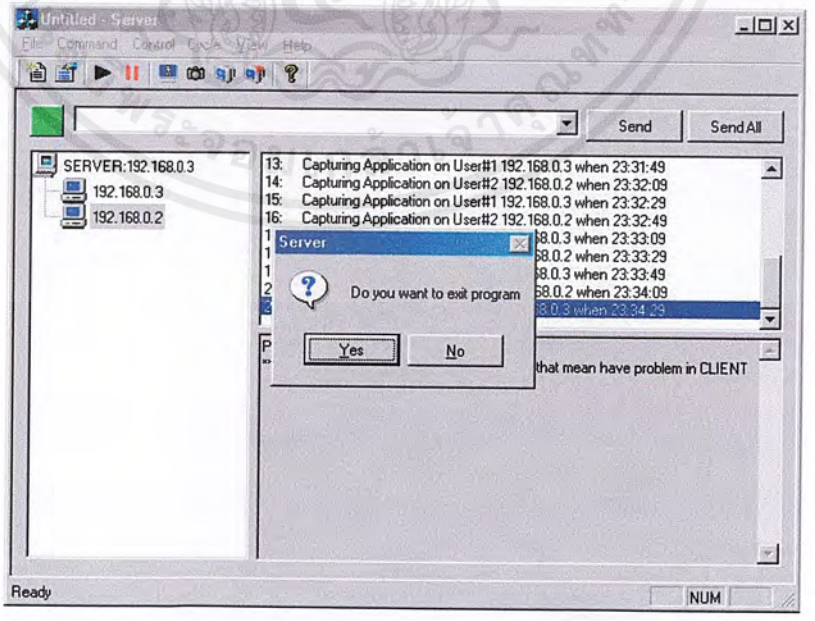
7.2.14 เมื่อผู้ใช้ต้องการปิดโปรแกรมเครื่องควบคุม มีขั้นตอนดังนี้

ขั้นตอนที่ 1 เลือกคำสั่ง Exit จากเมนูบาร์



รูปที่ 7.59 แสดงการเลือกคำสั่ง Exit จากเมนูบาร์ เพื่อปิดโปรแกรมเครื่องควบคุม

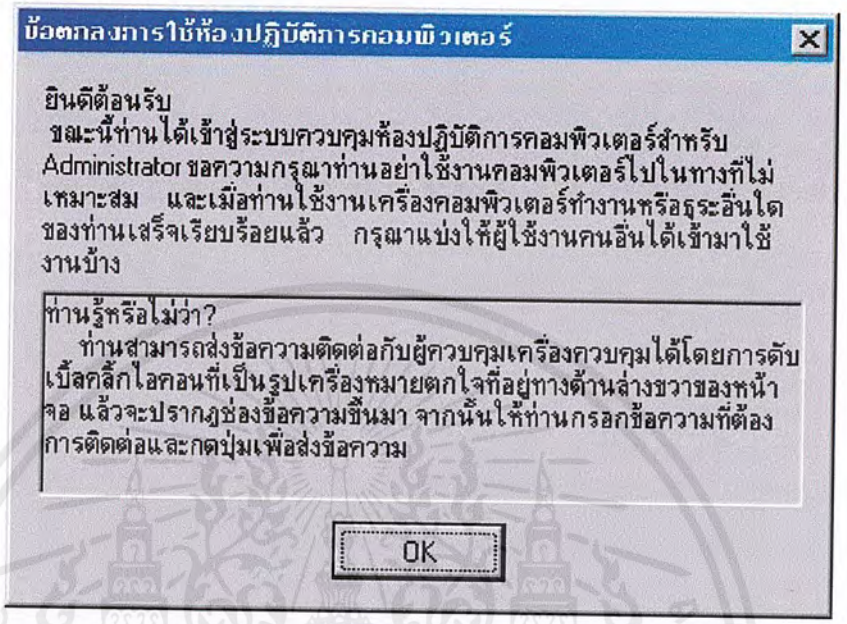
ขั้นตอนที่ 2 เมื่อเลือกคำสั่งดังกล่าวจะปรากฏกล่องข้อความเพื่อยืนยันคำสั่ง ดังนี้



รูปที่ 7.60 แสดงกล่องข้อความเพื่อยืนยันคำสั่งปิดโปรแกรมเครื่องควบคุม

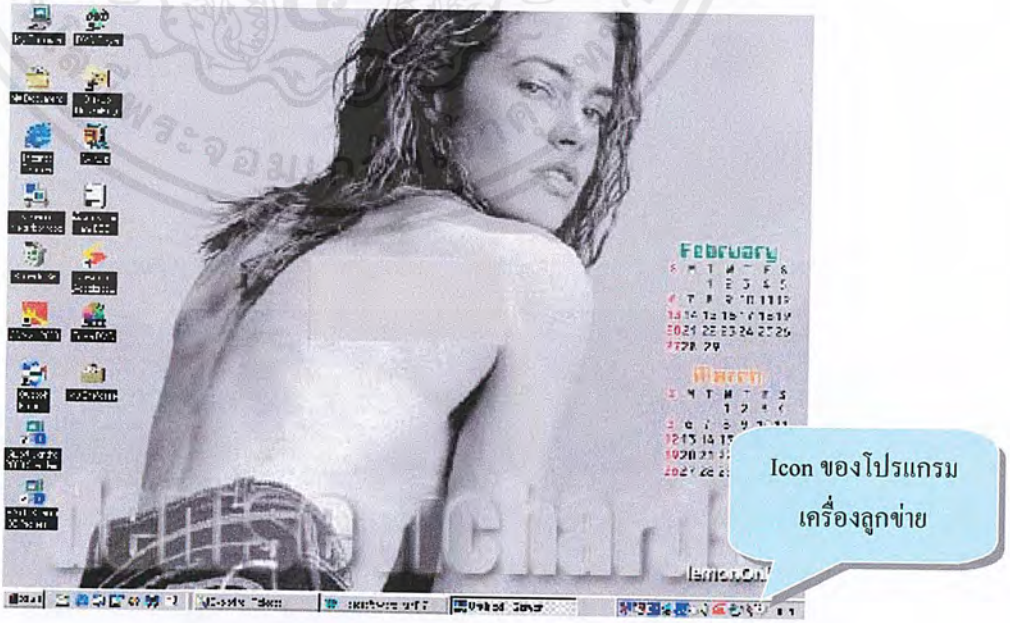
7.3 การทำงานหลักของโปรแกรมเครื่องลูกข่าย

7.3.1 เมื่อเครื่องลูกข่ายเข้าสู่ระบบจะปรากฏข้อตกลงการใช้ห้องปฏิบัติการคอมพิวเตอร์ ซึ่งมีข้อความดังนี้



รูปที่ 7.61 แสดงข้อตกลงการใช้ห้องปฏิบัติการคอมพิวเตอร์

7.3.2 เมื่อเครื่องลูกข่ายเข้าสู่ระบบจะปรากฏไอคอน (icon) ที่มุมด้านขวาของหน้าจอ ดังนี้

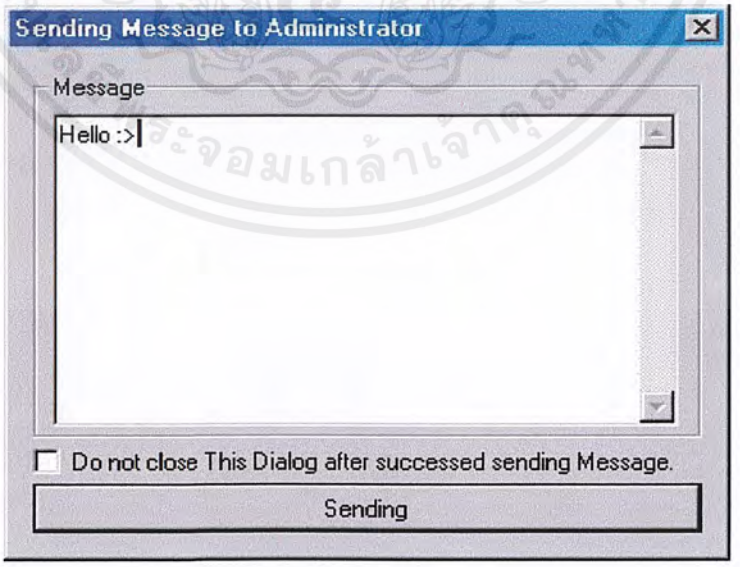


รูปที่ 7.62 แสดง Icon ของโปรแกรมเครื่องลูกข่ายที่มุมขวาของหน้าจอ

7.3.3 ขณะที่โปรแกรมเครื่องควบคุมตรวจจับแอพพลิเคชันของโปรแกรมเครื่องลูกข่ายอยู่นั้น โปรแกรมเครื่องลูกข่ายจะปรากฏหน้าจอดังนี้

รูปที่ 7.63 แสดงหน้าจอของเครื่องลูกข่าย ในขณะที่เครื่องควบคุมตรวจจับแอพพลิเคชันของเครื่องนั้น

7.3.4 เมื่อเครื่องลูกข่ายต้องการส่งข้อความไปที่เครื่องควบคุม ต้องดับเบิลคลิกที่ Icon แล้วจะปรากฏกล่องข้อความดังนี้



รูปที่ 7.64 แสดงกล่องข้อความที่เครื่องลูกข่ายสามารถส่งข้อความไปยังเครื่องควบคุมได้

7.3.5 เมื่อต้องการตัดการติดต่อกับเครื่องลูกข่ายออกจากระบบ สามารถทำได้ดังนี้

ขั้นตอนที่ 1 คลิกขวาที่ Icon จะปรากฏเมนูดังนี้

รูปที่ 7.65 แสดงการคลิกขวาที่ Icon

ขั้นตอนที่ 2 เลือกคำสั่ง Exit จะปรากฏกล่องข้อความให้พิมพ์ Password ดังนี้

รูปที่ 7.66 แสดงกล่องข้อความให้พิมพ์ Password ตัดการติดต่อกับเครื่องลูกข่าย

7.3.6 เมื่อเครื่องลูกข่ายออกจากระบบไม่ว่ากรณีใด ๆ จะปรากฏข้อความ Disconnect ที่เครื่องควบคุม

บทที่ 8
บทสรุปและข้อเสนอแนะ

8.1 บทสรุป

โปรแกรมที่ได้พัฒนาแบ่งเป็น 2 ส่วนคือ โปรแกรม PCLC-Server และโปรแกรม PCLC-Client โดย

8.1.1 โปรแกรม PCLC-Server มีความสามารถในการควบคุมการทำงานของเครื่องลูกข่ายโดยการส่งคำสั่ง (Message) ไปยังเครื่องลูกข่าย

8.1.2 โปรแกรม PCLC-Client มีความสามารถในการรับคำสั่งจากเครื่องควบคุมมาประมวลผลแล้วดำเนินการตามคำสั่งนั้น

โปรแกรมสามารถทำงานต่าง ๆ ตามลักษณะความสามารถที่ควรจะเป็น คือ

1) เครื่องควบคุมระบบที่ผู้ดูแลระบบใช้อยู่สามารถมองเห็นหน้าจอของเครื่องคอมพิวเตอร์ทุกเครื่องที่ต่ออยู่อย่างถูกต้องภายในระบบเครือข่ายของห้องปฏิบัติการคอมพิวเตอร์

2) สามารถส่งข้อความเตือนหรือโต้ตอบกันระหว่างเครื่องควบคุมระบบกับเครื่องคอมพิวเตอร์ลูกข่าย

3) สามารถตรวจจับและแสดงได้ว่าเครื่องคอมพิวเตอร์ลูกข่ายทุก ๆ เครื่องภายในห้องปฏิบัติการกำลังใช้งานแอพพลิเคชันใดอยู่บ้าง

4) ผู้ดูแลระบบสามารถสั่งปิดเครื่องคอมพิวเตอร์ลูกข่ายจากเครื่องควบคุมระบบได้ในกรณีที่เห็นว่าเหมาะสม

8.2 ข้อเสนอแนะ

8.2.1 ความละเอียดของหน้าจอเครื่องลูกข่ายที่อยู่ภายในระบบที่เหมาะสม ควรอยู่ไม่เกิน 800 x 600 และ บิตต่อพิกเซลไม่มากไปกว่า 16 บิต ซึ่งจะทำให้ขนาดไฟล์ BMP ที่ได้ ไม่ใหญ่มากจนเกินไป (จะอยู่ที่ประมาณ 1 เมกกะไบท์)

8.2.2 ควรปรับปรุงโปรแกรมให้สามารถทำงานได้ในลักษณะ Multi-Server หมายความว่าระบบสามารถควบคุมโดยผู้ดูแลระบบหลาย ๆ คนได้

8.2.3 ควรปรับปรุงให้มีระบบการ Log In เพื่อตรวจสอบผู้ที่เข้ามาใช้บริการได้

8.2.4 ควรมีระบบตรวจสอบแอพพลิเคชันที่ต้องห้ามหรือไม่ โดยอัตโนมัติแล้วแจ้งให้ผู้ดูแลระบบทราบ

8.2.5 การทำงานในลักษณะ Auto Operation ควรมีอัลกอริทึมในการตรวจสอบระยะเวลาในการทำงานให้เหมาะสมกับขนาดของเครือข่าย

8.2.6 ควรปรับปรุงให้สามารถทำงานได้บนระบบปฏิบัติการอื่น ๆ ได้ เช่น ลีนุกซ์

บรรณานุกรม

นิรุธ อำนวยศิลป์ คู่มือการเขียนโปรแกรม Microsoft Visual C++ Version 6.0

Barkakati Naba Visual C++ 2 Developer's Guide,Second Edition SAMS PUBLISHING 201 West 103rd Street Indianapolis,Indiana 46290

Hubbard John การเขียนโปรแกรมด้วยภาษา C++ แปลและเรียบเรียงโดย เบญจพร ศักดิ์ศิริ

Gregory Kate , et.al. Building Internet Application with Visual C++

ทรงเกียรติ ภาวดี , แกะรอย CGI บริษัท วิตตี้ กรุ๊ป จำกัด 52/3-4 เขต ป้อมปราบ กรุงเทพฯ 10100

Microsoft Win32 Programmer's Reference,Volume1

http://www.codeguru.com