การพัฒนาไคลเอนต์เซิร์ฟเวอร์บนสภาพแวดล้อมยูนิกซ์

CLIENT/SERVER DEVELOPMENT ON UNIX ENVIRONMENT

นายอัครเดช ศิริประภาพรชัย
นายอุดม รานอก

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2542

การพัฒนาไคลเอนต์เซิร์ฟเวอร์บนสภาพแวดล้อมยูนิกซ์

CLIENT/SERVER DEVELOPMENT ON UNIX ENVIRONMENT

โดย

นายอัครเดช ศิริประภาพรชัย

นายอุดม รานอก

อาจารย์ที่ปรึกษา

ผศ. บรรจง ปิยธำรง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปริญญานิพนธ์ปีการศึกษา 2542

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาไคลเอนต์เซิร์ฟเวอร์บนสภาพแวดล้อมยูนิกซ์

CLIENT/SERVER DEVELOPMENT ON UNIX ENVIRONMENT

ผู้จัดทำ

1. นายอัครเดช ศิริประภาพรชัย รหัสประจำตัว 40013280
2. นายอุดม รานอก รหัสประจำตัว 40013282

_____ อาจารย์ที่ปรึกษา

(ผศ. บรรจง ปิยธำรง)

การพัฒนาไคลเอนต์เซิร์ฟเวอร์บนสภาพแวดล้อมยูนิกซ์

นายอัครเดช ศิริประภาพรชัย 40013280
นายอุดม รานอก 40013282
ผศ. บรรจง ปิยธำรง อาจารย์ที่ปรึกษา
ปีการศึกษา 2542

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งในการนำเสนอการพัฒนาไคลเอนต์เซิร์ฟเวอร์บนสภาพแวดล้อมยูนิกซ์ เป็นการวิจัยเพื่อศึกษาการทำงานการประมวลผลแบบกระจาย ซึ่งรูปแบบในการพัฒนามีได้หลายรูปแบบ อาทิเช่น ซ็อคเก็ตซิสเต็มคอล (Socket System Call), ทรานสปอร์ตเลเยอร์อินเทอร์เฟซ (Transport Layer Interface), รีโมทโพรซีเดอร์คอล (Remote Procedure Call) เป็นต้น ในโครงงานนี้ได้ทำการพัฒนาโดยใช้รูปแบบของรีโมทโพรซีเดอร์คอล ทำการพัฒนาและทดสอบกลไกการทำงาน โดยจำลองการทำงานของตู้ ATM และหน่วยประมวลผลศูนย์กลาง ในการทำงานจะต้องควบคุมการใช้ทรัพยากรร่วมกันของแต่ละโปรเซส การทำมัลติเธรดดิ้ง (Multithreading) การทำโรลแบ็ค (Rollback) ข้อมูลเมื่อเกิดความผิดพลาด การควบคุมความถูกต้องของข้อมูลในการติดต่อสื่อสารผ่านระบบเครือข่าย

CLIENT/SERVER DEVELOPMENT ON UNIX ENVIRONMENT

Akaradech Siripraparpornchai

Udom Ranok

Asst.Prof. Banjong Piyatamrong Advisor

Year 1999

ABSTRACT

This thesis is present about Client/Server development on UNIX environment. Research for study distributed processing which the model of development such as Socket System Call, Transport Layer Interface and Remote Procedure Call. For this project using Remote Procedure Call test mechanical working. By using ATM and central processing simulation. In the work is control resource sharing of process, multithreading, rollback when the data is failure and control intrigrity of data communication between process on computer network, as the case study.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดีทางผู้จัดทำต้องขอขอบคุณบุคคลทุกท่านดังนี้

- บิดามารดาบุคคลที่สำคัญยิ่งของข้าพเจ้าที่ทำให้ข้าพเจ้าได้เกิดมา เป็นผู้ดูแลเอาใจใส่ในตัวของข้าพเจ้าตลอดจนคอยเป็นกำลังใจและคอยสนับสนุนในการศึกษาของข้าพเจ้าตลอดมา
- อาจารย์บรรจง ปิยธำรง ผู้เป็นที่ปรึกษาและให้ความรู้ต่างๆในการทำปริญญานิพนธ์นี้
- เพื่อนๆห้อง P และน้องๆห้อง D ในห้องคาต้าเบสทุกคน และเพื่อนๆที่สังกัดห้องแลปต่างๆที่คอยให้คำปรึกษา และคอยช่วยเหลือในการแก้ปัญหาต่างๆและเป็นกำลังใจในการทำงานตลอดจนความสนุกในการใช้ชีวิตอยู่ที่สถานศึกษาแห่งนี้
- สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังและอาจารย์คณะวิศวกรรมคอมพิวเตอร์ทุกท่าน ที่ซึ่งให้โอกาสข้าพเจ้าได้เข้ามาศึกษาและให้ความรู้แก่ข้าพเจ้าจนมีวันนี้

อัครเดช ศิริประภาพรชัย

อุดม รานอก

สารบัญ

หน้าที่

บทคัดย่อภาษาไทย		I
บทคัดย่อภาษาอังกฤษ		II
กิตติกรรมประกาศ		III
สารบัญ		IV
สารบัญตาราง		VII
สารบัญภาพ		VIII
บทที่	1 บทนำ	1
	1.1 วัตถุประสงค์	1
	1.2 ขอบเขตของโครงงาน	1
	1.3 ขั้นตอนการทำงาน	2
บทที่	2 ทฤษฎีและหลักการ	3
	2.1 ไคลเอนต์เซิร์ฟเวอร์	3
	2.2 Socket System Call	4
	2.2.1 โครงสร้างของซ็อคเก็ต	4
	2.2.2 ลักษณะซ็อคเก็ตและการนำไปใช้งาน	5
	2.2.3 Socket โดยใช้ Virtual-Circuit Service	7
	2.2.4 Socket โดยใช้ Datagram Service	8
	2.3 Transport Layer Interface(TLI)	9
	2.3.1 Synchronous and Asynchronous modes	10
	2.3.2 Transport Address	10
	2.3.3 Transport Service Data Units(TSDU)	10
	2.3.4 TLI Connection-Mode Service	11
	2.3.5 TLI Connectionless-mode service	12
	2.3.6 TLI Routines	13
	2.4 Windows Sockets	14
	2.4.1 Socket concept	15
	2.4.2 Client-Server Model	15
	2.4.3 Byte Ordering	15
	2.4.4 Winsock Network Model	16
	2.4.5 Socket Library Overview	17
	2.5 Remote Procedure Calls	19

2.5.1 โครงสร้างของ remote procedure call 19
2.5.2 การสร้าง RPC Program 21
2.5.3 การใช้ RPC compiler 21
2.5.4 การทำงานของ Remote Procedure Call 22
2.5.5 External Data Representation(XDR) 24
2.5.6 การเปรียบเทียบระหว่าง RPC และ OSI Model 24
2.6 การพัฒนาแอพพลิเคชั่นโดยใช้ RPC 25
2.6.1 การกำหนดโปรโตคอล 26
2.6.2 การใช้งาน RPCGEN 27
2.6.3 การกำหนดรูปแบบของภาษา (RPC Definition Language) 28
2.6.4 ฟังก์ชั่นของ ONC RPC ในระดับล่าง 33
2.6.5 การ Compile และการ Run โปรแกรม 35
2.6.6 สถานะของ Server มีความสำคัญอย่างไรและมีสถานะอะไรบ้าง 36
2.6.7 ความผิดพลาดที่เกิดขึ้น 36
2.7 ความรู้เรื่อง Automatic Transfer Machine 37
2.8 การพัฒนาแอพพลิเคชั่นบนยูนิกซ์เท็กซ์โหมด 37
2.8.1 curses 38
2.8.2 การใช้งาน curses 39
2.8.3 โครงสร้างพื้นฐานของโปรแกรม curses 41
บทที่ 3 การออกแบบและโครงสร้างของโปรแกรม 43
3.1 โครงสร้างการทำงานโดยรวมของระบบจำลอง ATM 43
3.2 โครงสร้างการทำงานของโปรแกรม 44
3.2.1 การออกแบบการติดต่อกันระหว่าง client และ server โดย RPC 44
3.2.2 โปรเซสและการอินเตอร์รัพโปรเซส 45
3.2.3 การทำ concurrency control ด้วยวิธี inetd 47
3.2.4 การทำ lock file สำหรับจัดสรรการใช้ทรัพยากร 50
3.2.5 การกำหนดเวลา time-out 51
3.2.6 การทำ Rollback 51
3.2.7 สัญญาณ (การอินเตอร์รัพโปรเซส) 53
3.3 โครงสร้างข้อมูล(Data Structure) 56
บทที่ 4 อัลกอริทึม 59
4.1 การส่งหมายเลขบัญชีและรหัสผ่านสำหรับตรวจสอบเพื่อขอใช้บริการ 59
4.2 การตรวจสอบหมายเลขบัญชีและรหัสผ่าน 60
4.3 การทำขอดูยอดคงเหลือ 61

4.4 การถอนเงิน 63
4.5 การโอนเงิน 64
บทที่ 5 การทดสอบและผลการทดสอบ 68
5.1 ทดสอบการทำ lockfile ของโปรแกรม 68
5.2 ผลหลังจากที่เข้าสู่โปรแกรม ATM Simulation 69
5.3 ผลการทำงานของโปรแกรมเมื่อมีการเรียกใช้บริการ การถอนเงิน 70
5.4 ผลการทำงานของโปรแกรมเมื่อมีการเรียกใช้บริการ ขอตรวจสอบยอดเงินคงเหลือ 71
5.5 ผลการทำงานของโปรแกรมเมื่อมีการเรียกใช้บริการ การโอนเงิน 72
บทที่ 6 สรุปและบทวิจารณ์ 73
บรรณานุกรม 74
ภาคผนวก ก 75
ภาคผนวก ข 77

สารบัญตาราง

ตาราง	หน้า
ตาราง 2.8.2-1 บางฟังก์ชั่นของ curses	41
ตาราง 2.8.2-2 ตารางแสดงแอททริบิวท์ใน curses	41

สารบัญรูป

รูป หน้าที่

รูปที่ 2.1.1-1 โครงสร้างข้อมูลของ Socket และ Descriptor Table 5

รูปที่ 2.2.2-1 Socket ทั้งสอง endpoint ในช่องสื่อสาร 5

รูปที่ 2.2.3-1 Socket แบบ Virtual-Circuit 8

รูปที่ 2.2.4-1 การทำงานของ Socket แบบ Datagram 9

รูปที่ 2.3-1 ลำดับการสื่อสารของแอพพลิเคชั่นแบบ Connection-oriented 12

รูปที่ 2.3-2 ลำดับการสื่อสารของแอพพลิเคชั่นแบบ Connectionless 13

รูปที่ 2.4-1 การเปรียบเทียบระหว่าง OSI Model และ Winsock Model 16

รูปที่ 2.4-2 ความสัมพันธ์ระหว่าง Application , WINSOCK.DLL และส่วนต่างๆของเน็ตเวิร์ก 19

รูปที่ 2.5.1-1 RPC setup 20

รูปที่ 2.5.4-1 การทำงานของ Remote Procedure Call (RPC) 23

รูปที่ 2.5.6-1 ตำแหน่งของ XDR และ RPC ที่อยู่ภายในมาตราฐาน OSI 25

รูปที่ 2.6.5-1 ขั้นตอนการคอมไพล์โดยใช้ rpcgen 35

รูปที่ 2.8.1-1 ปรแกรม curses 39

รูปที่ 3.1-1 แสดงโครงสร้างการทำงานโดยรวมของระบบจำลอง ATM ผ่านกลไก RPC 43

รูปที่ 3.2.1-1 การเรียกใช้ remote procedure ด้วยวิธี rpc 44

รูปที่ 3.2.2-1 ครงสร้างของโปรเซส 45

รูปที่ 3.2.3-1 การทำ mutiple client ด้วยวิธีของ inetd 47

รูปที่ 3.2.3-2 แสดงการทำงานของ inetd deamon 49

รูปที่ 3.2.4-1 การทำ lock file ในระบบ 50

รูปที่ 3.2.5-1 ส่วนของโปรแกรมที่กำหนดเวลา time-out 51

รูปที่ 3.2.6-1 การ interrupt signal สำหรับการทำ rollback 52

รูปที่ 4.1-1 การทำงานของโพรซีเดอร์ส่งหมายเลขบัญชีและรหัสผ่าน 59

รูปที่ 4.2-1 การทำงานของโพรซีเดอร์ ตวจสอบหมายเลขปัญชีและรหัสผ่าน 60

รูปที่ 4.3-1 การทำงานของโพรซีเดอร์ขอดูยอดคงเหลือ 62

รูปที่ 4.4-1 ทำงานของโพรซีเดอร์การถอนเงิน 63

รูปที่ 4-5-1 การทำงานของโพรซีเดอร์โอนเงิน(1) 66

รูปที่ 4.5-2 การทำงานของโพรซีเดอร์โอนเงิน(2) 67

รูปที่ 5.1-1 ลักษณะของโปรเซสที่กำลังอัพเดทไฟล์อยู่ 68

รูปที่ 5.1-2	การเข้าใช้งานอัพเดทไฟล์โดยไคลเอนต์	69
รูปที่ 5.2-1	หน้าจอแสดงเมื่อใส่ ID และรหัสผ่าน	69
รูปที่ 5.2-2	หน้าจอแสดงรายการต่างๆที่สามารถใช้บริการได้	70
รูปที่ 5.3-1	หน้าจอแสดงเมื่อเลือกบริการการถอนเงิน	70
รูปที่ 5.3-2	ผลลัพธ์หลังจากที่เซิร์ฟเวอร์ประมวลผลเสร็จแล้วส่งมายังฝั่งไคลเอ็นต์	71
รูปที่ 5.4-1	ผลลัพธ์ที่ได้จากโพรซีเดอร์ขอดูยอดคงเหลือ	71
รูปที่ 5.5-1	หน้าจอแสดงการขอใช้บริการโอนเงิน	72
รูปที่ 5.5-2	ผลลัพธ์จากการทำงานของโพรซีเดอร์โอนเงินแสดงที่ฝั่งไคลเอนต์	72

บทที่ 1
บทนำ

ในปัจจุบันนี้เครือข่ายคอมพิวเตอร์กำลังได้รับความนิยมและมีความสำคัญอย่างมากต่อการแข่งขันกันทางธุรกิจขององค์กร ดังนั้นการเลือกพัฒนาเครือข่ายที่เหมาะสมกับองค์กรจึงมีความจำเป็นมากสำหรับองค์กร ซึ่งจะช่วยในการลดค่าใช้จ่ายและใช้ทรัพยากรได้อย่างมีประสิทธิภาพ การพัฒนาการทำงานเช่นนี้มีหลากหลายรูปแบบ การพัฒนาแบบไคลเอนต์เซิร์ฟเวอร์ก็เป็นการทำงานอีกประเภทหนึ่งที่มีความน่าสนใจมาก มีลักษณะการจัดการเป็นแบบการกระจายข้อมูลในการทำงาน โดยจะแบ่งการทำงานออกเป็นสองส่วนคือส่วนที่ให้บริการ (Server) และส่วนที่ร้องขอการบริการ (Client) โดยที่ส่วนให้บริการจะทำการประมวลผลต่างๆตามที่ทางฝั่งร้องขอบริการเรียกใช้บริการ ซึ่งการทำงานเช่นนี้ภายในหนึ่งเซิร์ฟเวอร์อาจจะมีผู้มาร้องขอบริการได้หลายตัว หรืออาจจะมีผู้ให้บริการหลายๆตัวก็ได้ขึ้นอยู่กับการออกแบบและลักษณะงานที่จะนำไปใช้

1.1 วัตถุประสงค์

โครงงานนี้เป็นการศึกษาเพื่อออกแบบ และทดสอบการประมวลผลแบบกระจายการทำงาน และควบคุมความถูกต้องในการทำงาน ควบคุมการทำงานแต่ละโปรเซสให้ทำงานไปได้อย่างสอดคล้องกัน ซึ่งการทำงานในรูปแบบนี้มีได้หลายอย่างเช่น ซ็อคเก็ตซิสเต็มคอล(Socket System Call) หรือรีโมทโพรซีเดอร์คอล(Remote Procedure Call) เป็นต้น โดยที่โครงงานนี้จะใช้การจำลองการทำงานของการใช้ระบบ ATM ด้วยวิธีการแบบ Remote Procedure Call เป็นกรณีศึกษา อีกทั้งยังเป็นการศึกษาวิธีการทำงานและวิธีการรับส่งข้อมูลผ่านระบบเครือข่าย ลักษณะของแอพพลิเคชั่นนี้จะเป็นการทำงานของบนสภาพแวดล้อมในระบบยูนิกซ์ ซึ่งติดต่อผู้ใช้เป็นระบบเท็กซ์โหมด

1.2 ขอบเขตของโครงงาน

ปริญญานิพนธ์นี้จะศึกษาการทำงานของ Remote Procedure Call, Socket System Call ,Transport Layer Interface และ Winsock โดยในขั้นแรกจะทำการศึกษาถึงลักษณะกลไกในการทำงานของทุกรูปแบบว่ามีรูปแบบกับการทำงานประเภทไหน ซึ่งลักษณะที่จะศึกษานี้เป็นการพัฒนาระบบจำลองการใช้บริการตู้ ATM โดยให้ฝั่งหนึ่งทำงานเป็นตัวอินพุทข้อมูลและแสดงเอาท์พุทของข้อมูลที่เป็นผลลัพธ์โดยมีลักษณะการทำงานเป็นแบบไคลเอนต์ และทำการติดต่อไปยังอีกฝั่งคือเซิร์ฟเวอร์จะเปรียบเสมือนกับเป็นศูนย์กลางการประมวลผล ซึ่งภายในการทำงานของฝั่งประมวลผลจะมีการให้บริการที่สำคัญอยู่ 3 ลักษณะคือ บริการตรวจสอบยอดเงิน , บริการถอนเงิน และสุดท้ายคือบริการโอนเงินเข้าบัญชีผู้อื่น โดยการทำงานของฟังก์ชันทั้งหมดจะมาจากการรับค่าจากอินเทอร์เฟซต่างๆทางฝั่งไคลเอนต์หรือเหมือนกับรับค่าต่างๆจากตู้ ATM และนำค่าเหล่านั้นไปทำการประมวลผล และส่งผลลัพธ์ที่ได้กลับมายังฝั่งที่เปรียบเสมือนตู้ ATM นอกจากนี้ยังมีการแก้ไขปัญหาต่างๆที่อาจจะเกิดขึ้นระหว่างการทำงานได้ ยกตัวอย่างเช่น

การทำ locked file เพื่อจัดการเกี่ยวกับการครอบครองทรัพยากรในการทำงานเพื่อไม่ให้เกิดการอัพเดทข้อมูลผิดพลาด การทำ Rollback ของข้อมูลเมื่อเกิดการผิดพลาดในการสื่อสารขึ้นทำให้ข้อมูลที่ทำการส่งไปถือว่าไม่มีอะไรเกิดขึ้นเป็นต้น

1.3 ขั้นตอนการทำงาน

ขั้นตอนแรกเริ่มจากศึกษาหาข้อมูลเกี่ยวกับการทำงานแบบต่างๆของระบบไคลเอนต์เซิร์ฟเวอร์ เช่น Socket System Call, Transport Layer Interface, Remote Procedure Call และ WinSocks จากแหล่งต่างๆ ไม่ว่าจะเป็นห้องสมุดหรือตามเวปไซต์เป็นต้น จากนั้นทำการกำหนดขอบเขตของงานให้ชัดเจนว่าจะพัฒนาระบบไคลเอนต์เซิร์ฟเวอร์ด้วยกลไกแบบใด และกำหนด Case Study เพื่อจะทดสอบการทำงาน ซึ่งในโครงงานนี้จะใช้กลไกของ Remote Procedure Call เมื่อกำหนดขอบเขตได้จึงเริ่มพัฒนาแอพพลิเคชั่นโดยใช้ภาษาที่พัฒนาคือภาษา C บนระบบยูนิกซ์ โดยมองระบบการทำงานออกเป็นสองฝั่ง และออกแบบให้มีความสัมพันธ์กันอย่างเหมาะสม โดยมองทางฝั่งไคลเอนต์เปรียบเสมือนการทำงานของตู้ ATM และมองการทำงานที่ฝั่งเซิร์ฟเวอร์เปรียบเสมือนการทำงานของศูนย์กลางการประมวลผลและเชื่อมการทำงานของสองฝั่งเข้าด้วยกัน และตรวจสอบข้อผิดพลาดต่างๆที่เกิดขึ้นและทำการแก้ไขให้เรียบร้อย

บทที่ 2
ทฤษฎีและหลักการ

2.1 ไคลเอนต์เซิร์ฟเวอร์

ปัจจุบันระบบเครือข่ายได้เริ่มเข้ามาแทนที่ระบบใหญ่ๆอย่างเมนเฟรมและมินิคอมพิวเตอร์แล้ว เนื่องจากเป็นการคำนวณแบบกระจายศูนย์หรือ distributed processing ซึ่งสามารถแบ่งงานกันได้ ในขณะที่เครื่องเมนเฟรมและมินิคอมพิวเตอร์ส่วนใหญ่ยังใช้การคำนวณแบบ central processing หรือระบบการคำนวณจากส่วนกลาง

ในแบบแบบกระจายศูนย์มีหลายรูปแบบด้วยกันแต่ละแบบมีประสิทธิภาพที่แตกต่างกัน แต่แบบที่กำลังนิยมอยู่ในขณะนี้เรียกว่า Client/Server processing ซึ่งแบ่งออกได้สองชนิดคือ Shared-device processing และแบบ Client/Server processing

ระบบแบบ shared-device processing ถูกพัฒนาขึ้นเพื่อตอบสนองการขยายตัวของระบบเครือข่ายแลนระบบนี้มีข้อดีคือเร็ว แสดงดังรูป 2.1-1 การทำงานของแอพพลิเคชันจะวิ่งอยู่บนโฮสต์ ซึ่งในระบบแบบนี้จะเรียกโฮสต์ว่า เซิร์ฟเวอร์ การทำงานส่วนใหญ่ของมันจะยอมให้ผู้ใช้มาร่วมใช้รีซอร์สต่างๆที่อยู่บนเซิร์ฟเวอร์ได้ แต่รีซอร์สจะถูกจำกัดให้เป็นเพียงการร่วมกันใช้ไฟล์และเครื่องพิมพ์เท่านั้น

ข้อเสียของมันคือ การทำงานส่วนใหญ่จะอยู่ในเครื่องเวิร์กสเตชันของผู้ ใช้เท่านั้น ยกเว้นฟังก์ชั่นการพิมพ์และการทำไฟล์ I/O ที่จะอยู่บนเครื่องเซิร์ฟเวอร์ เมื่อผู้ใช้ต้องการใช้แอพพลิเคชันหรือไฟล์มันจะทำการพิมพ์และการร้องขอมาที่เซิร์ฟเวอร์ เซิร์ฟเวอร์จะส่งไฟล์ทั้งหมดหรือ แอฟพลิเคชันนั้นๆเข้ามาทำงานต่อไป เซิร์ฟเวอร์ที่ทำหน้าที่ในการแล่งไฟล์กันใช้นี้เรียกว่าไฟล์เซิร์ฟเวอร์(File server) ฟังก์ชันทางการพิมพ์ก็คล้ายๆกันคือ เมื่อผู้ใช้ต้องการพิมพ์ไฟล์ข้อมูล ก็จะส่งไฟล์นั้นไปให้เซิร์ฟเวอร์เพื่อส่งออกการพิมพ์ เซิร์ฟเวอร์ที่ทำหน้าที่นี้จะเรียกว่า Print server บางครั้งเซิร์ฟเวอร์ตัวเดียวสามารถทำหน้าที่ของ File server และ Printer server ได้ระบบเครือข่ายที่สามารถรองรับการทำงานแบบนี้ไจะต้องเป็นระบบที่มีความเร็วพอสมควร เพราะข้อมูลที่วิ่งไปมาในระบบมีขนาดใหญ่(ส่งรับข้อมูลทั้งไฟล์หรือดาต้าเบส) ระบบที่นิยมใช้โดยทั่วไปคือ Novell Netware และ Microsoft LAN Manager เป็นต้น ทั้งสองตัวจะวิ่งโดยมาตราฐานหลายอย่างรวมทั้ง Ethernet ที่มีความเร็ว 10 Mbps หรือ Token ring ที่มีความเร็ว 4 หรือ 16 Mbps

เซิร์ฟเวอร์ที่ใช้ในระบบไม่จำเป็นต้องมีขนาดซีพียูที่ใหญ่โตซึ่งมีราคาแพงอย่างในระบบเมนเฟรมและมินิคอมพิวเตอร์อีกต่อไปแล้ว แต่เครื่องเวิร์กสเตชันจำเป็นจะต้องมีซีพียูที่สามารถวิ่งแอพพลิเคชันของตนเองได้ความเร็วของการทำงานก็จะถูกจำกัดอยู่ในเครื่องเวิร์กสเตชันนั่นเอง สำหรับระบบ Client/Server processing จะคล้ายๆกับระบบ Shared-device processing มากแต่แทนที่การคำนวณทางด้านแอพพลิเคชันจะอยู่ในเครื่องเวิร์กสเตชันของผู้ใช้ทั้งหมด จะแบ่งการคำนวณมาอยู่ในตัวเซิร์ฟเวอร์ด้วย

จะเห็นว่าแอพพลิเคชันทำงานทั้งในเครื่องเซิร์ฟเวอร์และเวิร์กสเตชันด้วย เมื่อผู้ใช้ต้องการข้อมูลบางส่วนที่อยู่ในดาต้าเบสไฟล์ขนาดใหญ่ เครื่องเวิร์กสเตชันจะส่งคำร้องขอข้อมูลนั้นไปที่ตัวเซิร์ฟเวอร์ เซิร์ฟเวอร์จะทำการคำนวณและดึงข้อมูลเหล่านั้นออกมา และส่งเฉพาะข้อมูลนั้นๆกลับมายังเครื่องเวิร์กสเตชันที่ร้องขอมา เมื่อเวิร์กสเตชันได้รับข้อมูลมันก็จะนำมาคำนวณต่อไป

จะเห็นได้ว่าข้อมูลที่ส่งไปมาจะมีเพียงข้อมูลที่จำเป็นเท่านั้น ในขณะที่ถ้าส่งในระบบ Shared-device processing จะเป็นการส่งไฟล์ดาต้าเบสใหญ่ๆทั้งไฟล์มาที่เครื่องเวิร์กสเตชัน ทั้งๆที่มันต้องการข้อมูลเพียงไม่กี่ไบต์ก็ตาม เมื่อองค์กรได้มีการใช้แอพพลิเคชันแบบ Shared-device บนระบบเครือข่ายมากเกินไป จะทำให้ระบบเครือข่ายนั้นๆไม่สามารถรองรับ traffic ที่มากเช่นนั้นได้ เวลาในการตอบสนองของระบบต่อผู้ใช้ก็จะมีค่าสูงขึ้นด้วย แต่ถ้าหน่วยงานใดหันมาใช้แอพพลิเคชันแบบไคล์เอนต์เซิร์ฟเวอร์มากขึ้นก็ไม่จำเป็นต้องมากังวลถึง traffic ที่จะเกิดขึ้นในระบบเครือข่ายของคุณเลย เพราะระบบไคล์เอนต์เซิร์ฟเวอร์จะใช้ traffic น้อยมากเมื่อเทียบกับระบบแบบ Shared-device

ด้วยการส่งข้อมูลที่มีประสิทธิภาพแบบนี้ ทำให้การติดต่อกันข้าม WAN นี้สามารถทำได้รวดเร็วมากความสามารถและประสิทธิภาพในระบบก็มีสูงด้วย อีกทั้งแอพพลิเคชันที่วิ่งอยู่บนตัวเซิร์ฟเวอร์ที่คอยให้บริการทางด้านข้อมูลแก่ไคล์เอนต์ก็ไม่จำเป็นจะต้องเป็นอแพพลิเคชันแบบเดียวกัน แต่ขอให้มีมาตรฐานแบบเดียวกันก็พอ

2.2 Socket System Calls

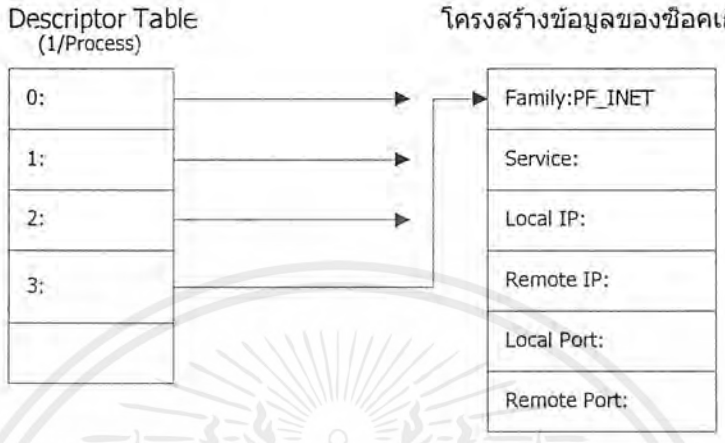
Socket System Call คือ กลุ่มของรูทีนที่ใช้ในการสร้างช่องการสื่อสารระหว่างแอพพลิเคชั่นบน Local System กับแอพพลิเคชั่นบน Remote System ซึ่งเป็นการติดต่อสื่อสารข้อมูลผ่านระบบเน็ตเวิร์ก ในระดับอินเทอร์เนตรูปแบบหนึ่ง ซึ่งมี BSD เป็นผู้ที่สร้างแล้วกำหนดมาตราฐานของซ็อคเก็ตโดยซ็อคเก็ตจะอ้างอิงกับ TCP/IP Protocol

ซ็อคเก็ตรูทีนจะคล้ายๆกับฟังก์ชั่นของ TLI ที่สามารถสร้างการสื่อสาร, จัดการการติดต่อ และส่งผ่านข้อมูล ตลอดจนสามารถเลือกใช้ Transport Provider ได้ทั้งแบบ Connectionless และ Connection-oriented

2.2.1 โครงสร้างของซ็อคเก็ต

ในยูนิกซ์เวลาที่แอพพลิเคชั่นมีการเรียกใช้ Open Function จะมีการสร้างไฟล์เดสคริปเตอร์(File Descriptor) ซึ่งใช้ในการเข้าถึงไฟล์(Access File) เก็บไว้ในตาราง(Descriptor Table)ในลักษณะหนึ่งตัวชี้ (Index Descriptor) จะเป็นตัวชี้ตำแหน่งภายในโครงสร้างข้อมูลของซ็อคเก็ต (Internal Data Structure Socket) จะมี Socket Descriptor ซึ่งเป็นตัวเลขแอดเดรสที่ชี้ไปยังตำแหน่งโครงสร้างข้อมูลของซ็อคเก็ตซึ่งมีลักษณะเหมือน File Descriptor ทุกครั้งที่แอพพลิเคชั่นเรียกใช้ซ็อคเก็ตจาก OS จะมีการสร้างโครงสร้างข้อมูล ขึ้นเพื่อใช้เก็บรายละเอียด(Information)เพื่อใช้ในการติดต่อสื่อสารและนำตำแหน่งของโครงสร้างข้อมูล ไปเก็บไว้ใน Descriptor Table ในลักษณะพอยเตอร์(Pointer) ดังตัวอย่างในรูป 2.1.1-1 ทุกครั้งที่มีการเรียกใช้คำสั่งสร้างซ็อคเก็ต (Create Socket) จะต้องการ Initiate Connection ถ้าเซิร์ฟเวอร์เรียกใช้

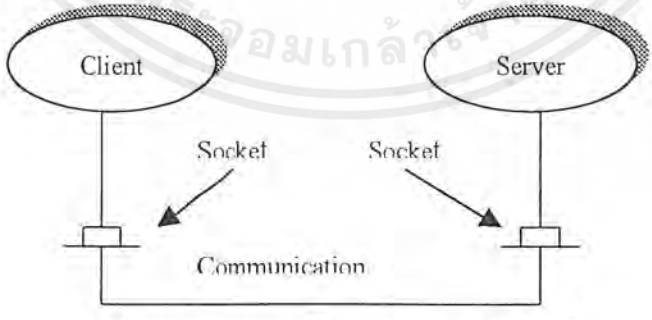
Socket จะเรียกว่าพาสซีฟซ็อคเก็ต (Passive Socket) ถ้าฝั่งไคลเอนต์ทำการ Initiate Connection จะเรียกว่าแอคทีฟซ็อคเก็ต(Active Socket) หลังจาก Initiate ถ้ามีการ Connect ตอบจะทำการเก็บ Information ลงบน Socket Data Structure ก่อนที่จะทำการใช้ซ็อคเก็ตจริง



รูปที่ 2.1.1-1 โครงสร้างข้อมูลของ Socket และ Descriptor Table

2.2.2 ลักษณะซ็อคเก็ตและการนำไปใช้งาน

Socket คือ end-point หนึ่งในช่องการสื่อสารทั้งสองฝั่ง ซึ่งสามารถใช้ Socket routine สร้างช่องสื่อสาร อาจจะเป็น Virtual circuit หรือ Datagram ก็ได้ และใช้ ช่องสื่อสารนี้ในการส่งข้อมูลระหว่าง Application ได้ จากรูปที่ 2.2.2-1 แสดงสองแอพพลิเคชั่น ใช้ซ็อคเก็ตโดยซ็อคเก็ตจะอยู่ที่จุดปลายของ Channel เมื่อสร้างซ็อคเก็ต จะส่งค่าไฟล์เดสคริปเตอร์ (File Descriptor) ที่จะถูกใช้เมื่อการ Access endpoint นั้น



รูปที่ 2.2.2-1 Socket ทั้งสอง endpoint ในช่องสื่อสาร

Socket System Call สามารถแบ่งออกเป็น 2 กลุ่มโดยกลุ่มแรกจะทำงานภายใต้ฟังก์ชั่นและยูทิลิตี้รูทีน(Utility Routines) อีกกลุ่มจะทำงานในลักษณะ Client/Server

การใช้ Socket System Call จะมีความซับซ้อนบ้างเพราะซ็อคเก็ตมีพารามิเตอร์ซึ่งสามารถนำไปใช้ในโปรแกรมได้หลายรูปแบบสามารถใช้ติดต่อสื่อสารระหว่างไคลเอนต์กับเซิร์ฟเวอร์ในลักษณะ TCP หรือ UDP ได้ด้วยการกำหนด Remote Endpoint Address แบบเฉพาะเจาะจงสำหรับไคลเอนต์และกำหนด Remote Endpoint Address แบบไม่เฉพาะเจาะจงสำหรับเซิร์ฟเวอร์ โดย Socket System Call มี 8 ฟังก์ชั่นพื้นฐานดังนี้

1. การสร้างซ็อคเก็ต จะใช้ **Socket() routine** ในการสร้าง Socket เมื่อใช้ในการสื่อสารเครือข่าย(Network Communication) โดยเริ่มเก็บ Information Socket ใหม่ลงบน Descriptors เช่น แอพพลิเคชั่นต้องการใช้โปรโตคอลแบบใดเช่น Protocol Family Internet หรือ PF-INET สำหรับ TCP/IP Service แบบ TCP หรือ UDP ถ้าใช้ซ็อคเก็ตในแบบ Internet Protocol Family

2. การติดต่อกับซ็อคเก็ตอื่น จะใช้ **Connect() routine** หลังจากที่เรียกใช้ Create Function และทางฝั่งไคลเอนต์จะทำการเรียก Connect Function เพื่อทำการติดต่อกับ Remote Server ซึ่งใน Connect Function ของไคลเอนต์จะระบุ IP address และ Protocol Port ของ Remote Machine ไว้หลังจากที่ทำการ Connect กับ Remote Machine แล้วจึงทำการส่งข้อมูลต่อไป

3. การส่งข้อมูลให้ซ็อคเก็ตอื่นจะใช้ **Write() routine** จะถูกใช้ในการส่งข้อมูลบน TCP Connection ระหว่างไคลเอนต์กับเซิร์ฟเวอร์โดยทางไคลเอนต์จะเรียกว่าจะส่งการร้องขอ(Send Request) ส่วนทางเซิร์ฟเวอร์จะเรียกว่าจะส่งการตอบรับ(Send Replies) Write function มีสาม Arguments ในการส่งผ่านข้อมูลคือ Descriptor ของ Socket, Address ของ Data และ Length ของข้อมูล โดยปกติแล้ว Write Function จะทำการสำลองข้อมูลจากบัฟเฟอร์ของเคอร์เนล OS แล้วส่งออกไปยังในระบบเครือข่าย ถ้าเกิด System Buffer เต็มแล้ว Write Function จะทำการหยุดส่งข้อมูลแต่ใน TCP จะทำการสร้างบัฟเฟอร์ เพิ่มเพื่อรองรับข้อมูลที่จะส่งเข้ามา

4. การอ่านข้อมูลจากซ็อคเก็ตจะใช้ **Read() routine** จะถูกใช้เป็นตัวรับข้อมูลระหว่างไคลเอนต์กับเซิร์ฟเวอร์ บน TCP Connection โดยปกติหลัง จาก การเกิดการติดต่อเรียบร้อยแล้ว ฝั่งเซิร์ฟเวอร์ จะมี Read Function ขอรับการติดต่อจากไคลเอนต์ โดย Write Function หลังจากที่ไคลเอนต์ได้ส่งสัญญาณการติดต่อออกไปแล้ว ฝั่งไคลเอนต์จะใช้ Read Function เพื่อรอรับการตอบรับจาก Server Read Function จะประกอบด้วยสามอาร์กิวเมนต์คือ Descriptor Address ของบัฟเฟอร์และขนาดของบัฟเฟอร์จะทำการขยายข้อมูลที่ได้รับมาแล้วนำข้อมูลที่ได้จากซ็อคเก็ตไปไว้ในบัฟเฟอร์ที่ใช้งานถ้าข้อมูลที่ได้รับมาไม่ครบก็จะส่งขนาดของข้อมูลที่พบกลับไปในไคลเอนต์และเซิร์ฟเวอร์ ค่าซ็อคเก็ตใช้ Read Function ได้รับข้อความ (Message) แบบ UDP แล้วจะไม่มีการนำข้อความมาเก็บไว้ในบัฟเฟอร์ แต่จะส่งไปให้ แอพพลิเคชั่นเลยแต่ถ้าเป็นการส่งแบบ connection-Oriented จะต้องมีสามอาร์กิวเมนต์คือ socket Descriptor, ที่อยู่ของข้อมูลในบัฟเฟอร์และขนาดของบัฟเฟอร์

5. การเลิกใช้ซ็อคเก็ตจะใช้ **Close() routine** ไคลเอนต์และเซิร์ฟเวอร์ที่ใช้เมื่อจบการใช้ ซ็อคเก็ตเพื่อจะทำการคืนค่าที่จองไว้(Deallocate) ถ้าซ็อคเก็ตนี้มีเพียงโปรเซสเดียวที่ใช้ก็จะทำการ Terminate และ Deallocate เมื่อเรียกใช้ Close Funcion แต่ถ้าเป็นการใช้ซ็อคเก็ตร่วมกันหลายๆโปรเซส โดย ทุก ๆ โปรเซส ต้องเรียกใช้ Close Function เมื่อครบแล้วจึงทำการ Terminate และ Deallocate Socket นั้น

6. การหาแอดเดรสให้ซ็อคเก็ตจะใช้ **Bind() routine** ในตอนแรกที่เรียกใช้ซ็อคเก็จนั้น ซ็อคเก็ตยังไม่ถูกกำหนด Endpoint Address ของ Local และ Remote Application ต้องใช้ Bind Function บอก Local Endpoint Address ให้กับซ็อคเก็ต โดยผ่าน Socket Descriptor Argument และ Endpoint Address Argument สำหรับ TCP/IP Protocol Endpoint Address จะใช้ซ็อคเก็ตแอสเดรสอินเทอร์เน็ต(SocketAddr-In) Structure และ ต้องกำหนด IP และ Port ให้กับ Socket ด้วย ส่วนซ็อคเก็ตที่เซิร์ฟเวอร์ใช้ Bind Function เพื่อบอกว่าใช้พอร์ตอะไรเพื่อขอการติดต่อ

7. ใช้ **Listen() routine** เมื่อซ็อคเก็ตพร้อมที่จะทำงาน หลังจากที่เกิดการติดต่อ ระหว่างเซิร์ฟเวอร์และไคลเอนต์แล้ว ถ้าเปิด Connection-oriented ในเซิร์ฟเวอร์ใน Passive Model จะหมายถึงการรอรับการตอบรับการสื่อสารที่เข้ามาของไคลเอนต์และจะรอรับตลอดเวลาจนกว่าจะมีการติดต่อเข้ามา เซิร์ฟเวอร์จะทำการตอบรับในเวลาไม่กี่เสี้ยววินาที และถ้าการติดต่อสื่อสารขาดหายไป เซิร์ฟเวอร์ จะต้องใช้ Listen Function จะขนาดของ Queue โดยผ่าน Argument ของ Function ไปยัง OS เพื่อทำการติดต่อสื่อสารซ็อคเก็ตใหม่เพื่อเข้าสู่ Passive Mode

8. รอรับการติดต่อจากไคลเอนต์จะใช้ **Accept () routine** หลังจากที่ Socket () ได้ทำการ Bind และเข้าสู่ Passive Mode Server จะใช้ Accept Function สร้าง Connection เพื่อตอบสนองการติดต่อของไคลเอนต์ที่เข้ามาใหม่ Accept function จะสร้างซ็อคเก็ตขึ้นมาใหม่เพื่อรับการติดต่อจากไคลเอนต์ โดยจะส่ง Descriptor ของซ็อคเก็ตอันใหม่ออกไปและทำการส่งข้อมูลไปยังซ็อคเก็ตตัวใหม่ หลังจากที่จบการใช้ซ็อคเก็ตแล้ว เซิร์ฟเวอร์จะทำการปิดซ็อคเก็ต

2.2.3 Socket โดยใช้ Virtual-Circuit Service

การใช้ซ็อคเก็ตบนช่องสื่อสารที่เป็นแบบ Virtual-circuit นั้นคล้ายกับการใช้ connection-mode service ของ TLI จากรูปที่ 2.2.3-1 แสดงการทำงานของซ็อคเก็ตแบบ Virtual-Circuit Service
ฝั่งเซิร์ฟเวอร์ ต้องกระทำดังนี้

1. สร้างซ็อคเก็ต จะใช้ฟังก์ชั่น *socket()* ต้องระบุ address family, transport protocol และส่วนที่แสดงถึงการใช้ Virtual circuit service โดยฟังก์ชั่น *socket ()* จะส่ง file descriptor ที่สัมพันธ์กับ Socket นั้น

2. Bind transport address ไปที่ซ็อคเก็ต โดยใช้ฟังก์ชั่น *bind ()* เพื่อหาแอดเดรสที่อยู่บน Transport provider

3. จากนั้นจะนำซ็อคเก็ตที่ได้ให้อยู่ "listen state" โดยการเรียกฟังก์ชั่น *listen()*

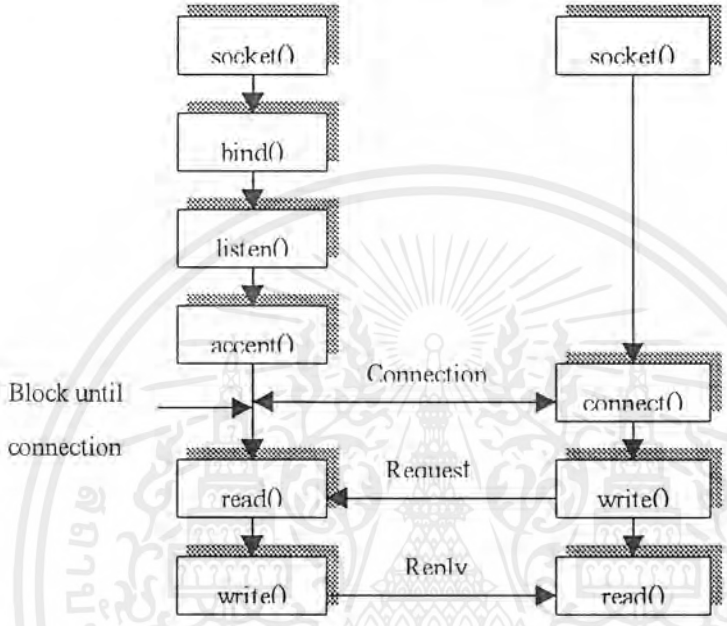
4. รอคอยการติดต่อจากไคลเอนต์โดยใช้ ฟังก์ชั่น *accept()* ฟังก์ชั่นนี้จะ sleep และจะ return ค่าเมื่อ Request จากไคลเอนต์มาถึง

ส่วนฝั่งไคลเอนต์ต้องกระทำงานดังนี้

1. สร้างซ็อคเก็ตโดยใช้ฟังก์ชั่น *Socket ()* เหมือนเซิร์ฟเวอร์ที่ต้องระบุ address family, transport protocol และส่วนที่แสดงถึงการใช้ virtual circuit service โดยฟังก์ชั่น *socket()* จะส่ง file descriptor ที่สัมพันธ์กับซ็อคเก็ตนั้น

2. Bind transport address ไปที่ซ็อคเก็ตโดยใช้ฟังก์ชั่น *bind()* เพื่อหาแอดเดรสที่อยู่บน transport provider ถ้าไม่ทำในขั้นตอนนี้ SVR4 จะ bind ให้โดยอัตโนมัติเมื่อติดต่อฝั่งเซิร์ฟเวอร์

3. สร้างการติดต่อกับเครื่องเซิร์ฟเวอร์ โดยใช้ฟังก์ชั่น *connect()* ซึ่งจะต้องระบุ transport address ของฝั่งเซิร์ฟเวอร์ไว้ด้วย หลังจากที่สร้างการติดต่อระหว่างไคลเอนต์และเซิร์ฟเวอร์ได้แล้ว สามารถที่จะทำการส่งข้อมูลระหว่างสองฝั่งได้ โดยเรียกใช้ฟังก์ชั่น *write()* สำหรับส่งข้อมูลจากซ็อคเก็ตหนึ่งไปซ็อคเก็ตอื่นและ *read()* สำหรับรับข้อมูลจากซ็อคเก็ตอื่นที่ส่งมา



รูปที่ 2.2.3-1 Socket แบบ Virtual-Circuit

2.2.4 Socket โดยใช้ Datagram Service

การใช้ซ็อคเก็ตบนช่องสื่อสารที่ใช้บริการแบบดาต้าแกรมจะเหมือนกับการส่งจดหมายโดยฝั่งไคลเอนต์เหมือนจะกระทำเสมือนเป็นบุรุษไปรษณีย์ ส่วนทางฝั่งเซิร์ฟเวอร์กระทำเสมือนเป็นผู้รับจดหมาย ซึ่งทั้งสองฝั่งจะต้องมีตู้ไปรษณีย์สำหรับการติดต่อนั่นก็คือซ็อคเก็ตนั่นเอง จากรูปที่ 2..2.4-1 แสดงการทำงานของซ็อคเก็ตแบบ Datagram Service

ฝั่งเซิร์ฟเวอร์มีการกระทำดังนี้

1. สร้างซ็อคเก็ต จะใช้ฟังก์ชั่น *socket()* โดยจะส่ง file descriptor ที่สัมพันธ์กับซ็อคเก็ตนั้น

2. Bind transport address ไปที่ซ็อคเก็ตโดยใช้ฟังก์ชั่น *bind()* เพื่อหาแอดเดรสที่อยู่บน Transport provider ซึ่งแอดเดรสจะใช้แอดเดรสนี้ในการส่งดาต้าแกรม

3. รอคอยดาต้าแกรมเดินทางมาถึง โดยใช้ฟังก์ชั่น *recvfrom()* ซึ่งจะ return ข้อมูลและแอดเดรสของผู้ส่งเมื่อข้อความข่าวสารเดินทางมาถึง

4. การ reply กลับจะใช้ฟังก์ชั่น *sento()* โดยต้องกำหนดแอดเดรสปลายทางและดาต้าแกรมที่ต้องการส่ง

ส่วนฝั่งไคลเอนต์มีการกระทำดังนี้

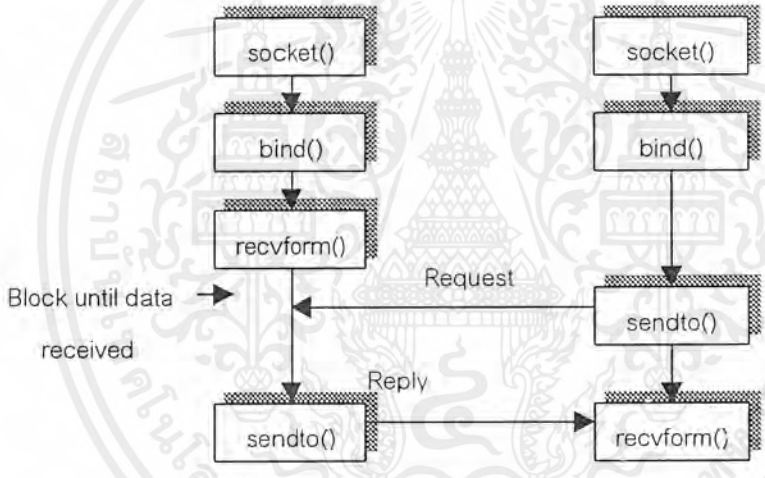
1. สร้างซ็อคเก็ตโดยใช้ฟังก์ชั่น *socket()* ซึ่งฟังก์ชั่น *socket()* จะส่ง file descriptor เข้าไปใน Transport provider

2. Bind transport address ไปที่ซ็อคเก็ตโดยใช้ฟังก์ชั่น *bind()* เพื่อหาแอดเดรสที่อยู่บน Transport provider ถ้าไม่ทำในขั้นตอนนี้ SVR4 จะ bind ให้โดยอัตโนมัติเมื่อติดต่อฝั่งเซิร์ฟเวอร์

3. สร้างการติดต่อกับเครื่องเซิร์ฟเวอร์ โดยใช้ฟังก์ชั่น *connect()* การใช้ฟังก์ชั่นนี้ไม่สามารถทำการติดต่อได้ แต่ต้องสัมพันธ์กับซ็อคเก็ต ที่ติดต่อและแอดเดรสปลายทาง

4. ส่งแมสเซสไปฝั่งเซิร์ฟเวอร์โดยใช้ฟังก์ชั่น *sento()* ซึ่งฟังก์ชั่นนี้ต้องการแอดเดรสของซ็อคเก็ตที่ต้องการจะส่งไป แต่ถ้าใช้ฟังก์ชั่น *Connect ()* ในการติดต่อกับ remote socket การส่งดาต้าแกรมนั้นสามารถที่จะใช้ฟังก์ชั่น *write ()* และ *read ()* ได้

5. ในการรับดาต้าแกรม ถ้าคาดหมายการ response ก็ให้ใช้ฟังก์ชั่น recvfrom() แต่ถ้ามีการติดต่อกับ Remote socket ได้แล้วให้ใช้ฟังก์ชั่น read () หรือ recv () จะดีกว่า



รูปที่ 2.2.4-1 การทำงานของ Socket แบบ Datagram

2.3 Transport Layer Interface(TLI)

TLI เป็น set หรือกลุ่มของซับรูทีนโปรแกรมมิ่งที่ใช้อินเทอร์เฟซโดยตรงกับ Transport Provider และ Transport user ที่สามารถใช้ในการติดต่อและส่งผ่านข้อมูลซึ่งกันและกัน จะใช้รูทีนเหล่านี้ในการสร้างแอพพลิเคชั่นทางด้านเน็ตเวิร์ก รูทีนของ TLI จะถูกออกแบบให้ทำงานกับ Transport Provider เช่น สามารถที่จะใช้ TLI ในการอินเทอร์เฟซด้วยโปรโตคอล TCP หรือ UDP โดยรูทีนเหล่านี้จะสามารถใช้ได้ทั้งรูปแบบ Connection-oriented และ Connectionless

TLI จะเป็นการทำงานระดับต่ำระหว่างเน็ตเวิร์กแอพพลิเคชั่น ซึ่งในแบบ Connection-oriented จะสามารถรับประกันความถูกต้องของข้อมูลว่ามีการส่งถึงปลายทางได้โดยถูกต้องไม่ผิดพลาด เพราะการทำงานแบบนี้จะมีการสร้างเส้นทางหรือเราเรียกว่า Virtual Circuit ของการติดต่อสื่อสารขึ้นมาก่อนการส่ง

ข้อมูล แต่ในแบบ Connectionless จะไม่รับประกันความถูกต้องของข้อมูลที่ปลายทางว่าจะถูกต้องหรือไม่ การทำงานของไคลเอนต์และเซิร์ฟเวอร์ของเน็ตเวิร์กแอพพลิเคชั่นจะต้องรู้จักส่วนที่เรียกว่า Transport Endpoint ในแบบ Connection-oriented เช่นการติดต่อกันโดยใช้โทรศัพท์ ซึ่งจุดเอ็นด์พอยต์ก็คือ โทรศัพท์เป็นต้น ส่วนแบบ Connectionless เช่นการส่งจดหมายที่ตู้ไปรษณีย์ จุด Endpoint ก็คือกล่องรับจดหมายนั่นเอง รูทีนของ TLI จะยอมให้เราจัดการเกี่ยวกับ Transport Endpoint ถ้าเป็นแบบ Connection-oriented รูทีนนี้จะยอมให้จับ Transport Address ที่เป็นจุดเอ็นด์พอยต์และแบบ Connectionless จะยอมให้จัดการการส่งและรับข้อมูลระหว่างจุดเอ็นด์พอยต์ได้

TLI จะใช้สร้างแอพพลิเคชั่นที่มีการควบคุมข้าม Transport Provider สามารถจัดการติดต่อกับเน็ตเวิร์กหลายๆเน็ตเวิร์กได้พร้อมๆกันและยังใช้เป็นตัวตรวจจับเหตุการณ์บน Transport Provider ได้ สามารถใช้สร้างแอพพลิเคชั่นที่ไม่สามารถใช้กลไกแบบ Remote Procedure Call ได้เพราะว่าบางแอพพลิเคชั่นไม่เหมาะกับการใช้รูปแบบของ RPC เช่น แอพพลิเคชั่นที่เกี่ยวกับ Remote Login ซึ่งจะต้องคอยดูช่องการติดต่อสื่อสารระหว่างไคลเอ็นต์และเซิร์ฟเวอร์ ซึ่งจะต้องมีการแลกเปลี่ยนส่งข้อมูลต่างๆกันตลอดเวลา เป็นต้น

2.3.1 Synchronous and Asynchronous modes

วิธีที่ใช้ในการรันแอพพลิเคชั่นมีอยู่สองรูปแบบคือ Synchronous mode หรือ asynchronous mode จะสามารถระบุชนิดของรูปแบบเมื่อใช้รูทีน t_open ()

Synchronous mode การทำงานของรูทีนจะไม่ทำการส่งค่ากลับมาเมื่อการทำงานของรูทีนนั้นยังไม่สำเร็จ เช่น การเรียกใช้รูทีน t_connect เพื่อสร้างการติดต่อ มันจะไม่ส่งค่ากลับมาเมื่อการสร้างเส้นทางการติดต่อยังไม่สำเร็จ แต่ใน Asynchonous mode รูทีนนั้นจะส่งค่ากลับมาก่อนที่การทำงานแต่ละรูทีนจะเสร็จสิ้น เช่นการเรียกใช้รูทีน t_connect() ของฝั่งไคลเอ็นต์มันจะทำการส่งค่ากลับมา เมื่อไหร่ที่การติดต่อมาถึง ด้านไคลเอ็นต์ของแอพพลิเคชั่นจะใช้รูทีน t_rcvconnect() บอกว่าการติดต่อสำเร็จ

2.3.2 Transport Address

TLI เป็นการออกแบบให้ทำงานกับ Transport Provider ใน Transport Provider ที่ต่างกันจะมีรูปแบบของแอดเดรสที่ต่างกัน โดยที่ตัว TLI จะยอมให้ระบุชนิดของแอดเดรสใน TLI จะกำหนดโครงสร้างทั่วไปเรียกว่า netbuf ซึ่งภายในจะประกอบด้วยชนิดของข้อมูลรวมถึง Transport Address โครงสร้างของ netbuf จะประกอบด้วย buf เป็นบัฟเฟอร์ที่เก็บลำดับของไบต์ ส่วน len จะระบุจำนวนของไบต์ในบัฟเฟอร์และ maxlen จะระบุจำนวนสูงสุดของไบต์ที่บัฟเฟอร์สามารถมี เมื่อไหร่ที่โครงสร้างนี้บรรจุ Transport Address ส่วนของ buf จะบรรจุแอดเดรส ส่วนของ len จะระบุจำนวนไบต์ในแอดเดรสและส่วนของ maxlen จะไม่ใช้

2.3.3 Transport Service Data Units(TSDU)

เป็นส่วนที่เรียกว่า TLI Support ซึ่ง TSDU นี้เป็นหน่วยของข้อมูลที่ด้านหนึ่งของแอพพลิเคชั่นส่งไปยังที่อื่นๆบน Transport Provider แบบคอนเน็คชั่นเลสจำนวนสูงสุดของ TSDU จะเป็นจำนวนสูงสุดของไบต์ที่สามารถส่งในดาต้าแกรม ส่วนแบบคอนเน็คชั่นโอเรียนเต็ด จำนวนสูงสุดของ TSDU จะเป็นจำนวนสูงสุดของไบต์ในข้อความหนึ่งๆ Message Unit เป็นชิ้นของข้อมูลที่จะข้ามไปในเน็ตเวิร์ก

2.3.4 TLI Connection-Mode Service

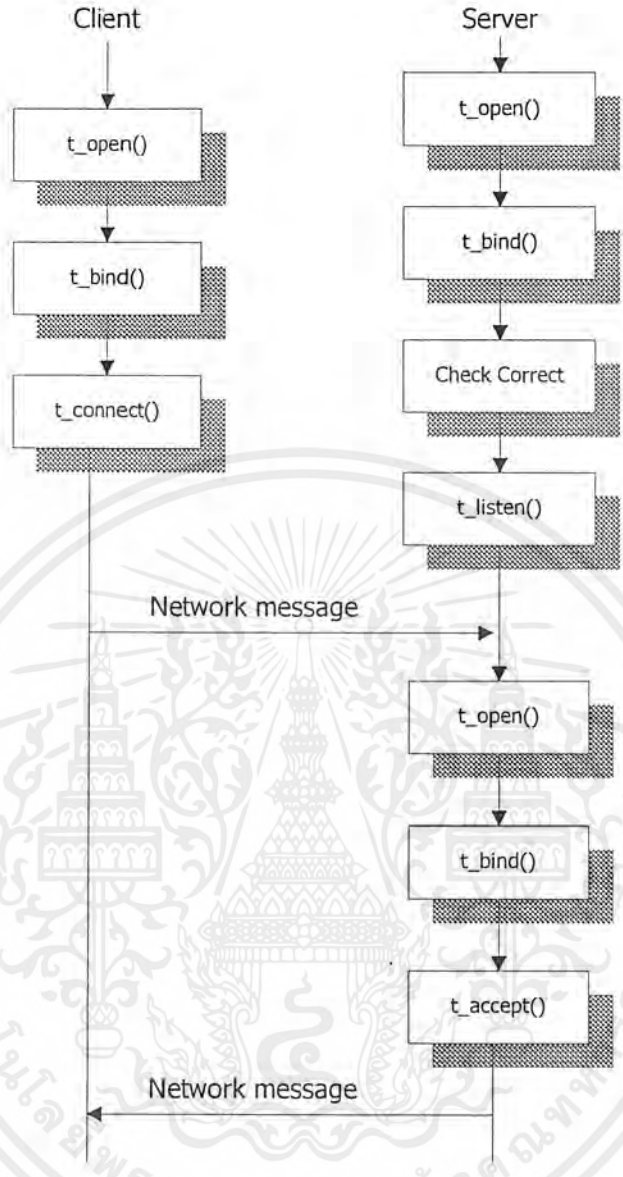
1. ทำการเปิด Clone device ที่มีลักษณะเหมือนกับ Transport Provider โดยใช้รูทีน t_open() โดยจะส่งไฟล์เดสคริปเตอร์ภายใน Transport Provider ซึ่งก็คือ Transport Endpoint นั่นเอง
2. ทำการบายด์ Transport Address ไปยัง Transport endpoint โดยใช้รูทีน t_bind() รูปแบบของ Transport address ขึ้นอยู่กับ Transport provider ที่ใช้
3. ตรวจสอบที่แอดเดรสที่ถูกถามโดยที่ TLI รูทีนจะส่งค่า unused address ถ้าแอดเดรสนั้นถูกใช้อยู่ ถ้าไม่ทำการบายด์แอดเดรสที่ถูกถามจะมีทางเลือกสองทางคือ
   - ออกจากแอพพลิเคชั่นแล้วทำการแสดงข้อผิดพลาดออกมา
   - ใช้แอดเดรสใหม่
4. t_listen() จะเป็นรูทีนที่รอคอยการร้องขอที่เข้ามา เมื่อไหร่ที่ต้องการร้องขอการติดต่อเข้ามาถึงรูทีนนี้จะรีเทิร์นและจะดำเนินการดังนี้
   - รับ Transport endpoint อื่นๆโดยใช้รูทีน t_open() อีกครั้ง
   - ทำการบายด์ Transport address ไปยัง Transport endpoint ตัวใหม่โดยใช้รูทีน t_bind()
   - ตอบรับการร้องขอการติดต่อโดยรูทีน t_accept() โดยจะผ่านค่า Transport endpoint ไปยังรูทีน t_accept() ซึ่งจะตอบรับการร้องขอบน Transport endpoint ที่สองเมื่อไหร่ที่รูทีนนี้ส่งค่า Transport endpoint จะสามารถรอรับการมาของการร้องขอครั้งต่อไปได้

ทางด้านฝั่งไคลเอนต์จะต้องดำเนินการตามขั้นตอนดังนี้

1. เปิด Clone device ที่เหมือนกับ Transport provider โดยใช้รูทีน t_open() จากนั้นจะรีเทิร์นค่าไฟล์เดสคริปเตอร์กลับมา
2. ทำการบายด์ Transport address ไปยัง Transport endpoint โดยใช้รูทีน t_bind()
3. สร้างการติดต่อกับเซิร์ฟเวอร์โดยใช้รูทีน t_connect() จะต้องระบุ Transport address ของด้านเซิร์ฟเวอร์ของแอพพลิเคชั่น

เมื่อทำการเชื่อมต่อระหว่างสองทรานสปอร์ตได้แล้วจะใช้รูทีนเหล่านี้ในการทำงาน

เมื่อต้องการที่จะส่งข้อมูลจาก Transport user ไปยังทรานสปอร์ตอื่นๆจะใช้รูทีน t_snd() ในการส่งข้อมูลเช่นเดียวกันเราจะใช้รูทีน t_rcv() ในการอ่านข้อมูลที่ส่งมาโดย peer transport user



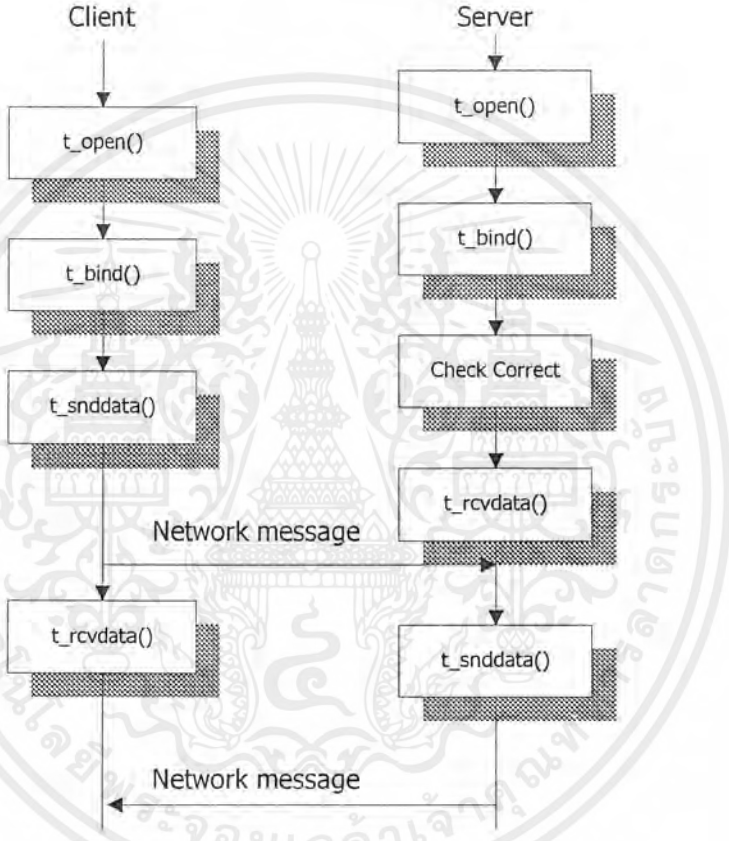
รูปที่ 2.3-1 ลำดับการสื่อสารของแอพพลิเคชั่นแบบ Connection-oriented

2.3.5 TLI Connectionless-mode service

1. เปิด Clone device โดยใช้รูทีน t_open() เมื่อกระทำแล้วจะได้ค่ารีเทิร์นกลับมาเป็นค่าไฟล์เดสคริปเตอร์ภายใน Transport provider ซึ่งไฟล์เดสคริปเตอร์เป็น Transport endpoint
2. รวม Transport address ไปยัง Transport endpoint โดยใช้รูทีน t_bind
3. ต่อมาทำการตรวจสอบแอดเดรสที่ร้องขอมาถูกใช้อยู่หรือไม่
4. คอยดาต้าแกรมที่มาถึงโดยใช้รูทีน t_rcvudata() เมื่อไหร่ที่ข้อความมาถึง รูทีนนี้จะส่งค่าข้อมูลและ Transport address ของผู้ส่ง
5. ถ้าต้องการทำการส่งข้อมูลให้ใช้รูทีน t_sndudata() จะต้องทำการระบุแอดเดรสปลายทาง

ด้านการทำงานของไคลเอ็นต์ ทำตามขั้นตอนดังนี้

1. เช่นเดียวกับฝั่งเซิร์ฟเวอร์คือต้องมีการเปิด Clone device โดยใช้รูทีน t_open() ซึ่งจะส่งค่าไฟล์เดสคริปเตอร์กลับมา
2. ทำการบายด์ Transport address ไปยัง Transport endpoint
3. ใช้รูทีน t_sndudata() ส่งข้อความไปยังเครื่องเซิร์ฟเวอร์
4. ใช้รูทีน t_rcvudata() ในการรอดาต้าแกรมที่ทำการส่งกลับมาแต่บางกรณีอาจจะไม่มีดาต้าแกรมส่งกลับมาเลยก็ได้ และการทำงานของคอนเน็คชั่นเลสไม่รับประกันความถูกต้องในการส่งข้อมูล



รูปที่ 2.3-2 ลำดับการสื่อสารของแอพพลิเคชั่นแบบ Connectionless

2.3.6 TLI Routines

รูทีนที่จำเป็นที่ใช้ในการจัดการข้าม Transport provider สามารถสรุปได้ดังนี้

รูทีนที่ใช้ในการจัดการทั่วไป

t_open()	สร้าง Transport endpoint และส่งค่าไฟล์เดสคริปเตอร์กลับมา
t_bind()	กำหนด Transport address ของเอ็นด์พอยต์
t_unbind()	ยกเลิก Transport address จากเอ็นด์พอยต์
t_close()	ทำลายเอ็นด์พอยต์
t_optmgmt()	ออปชั่นที่ใช้ในการเจรจากับ Transport provider

t_alloc()	จองหน่วยความจำสำหรับโครงสร้าง TLI
t_error()	แสดงข้อความผิดพลาด
t_free()	ยกเลิกการจองหน่วยความจำ
t_look()	รับเหตุการณ์ปัจจุบันบนเอ็นด์พอยต์

รูทีนที่ใช้แบบคอนเน็คชั่นโอเรียนเต็ด

t_connect()	ติดต่อไปยังรีโมททรานสปอร์ตยูสเซอร์
t_rcvconnect()	ตอบรับการติดต่อ
t_listen()	รอรับการร้องขอที่มาถึง
t_accept()	ตอบรับการร้องขอ
t_snd()	ส่งข้อมูล
t_rcv()	รับข้อมูล
t_snddis()	ส่งการร้องขอการยกเลิกการติดต่อ
t_rcvdis()	รับการร้องขอการยกเลิกการติดต่อ
t_sndrel()	ส่งการยกเลิกการร้องขอ
t_rcvrel()	รับการยกเลิกการร้องขอ

รูทีนที่ใช้แบบคอนเน็คชั่นเลส

t_sndudata()	ส่งข้อมูลดาต้าแกรม
t_rcvudata()	รับข้อมูลดาต้าแกรม
t_revuderr()	อ่านข้อมูลเกี่ยวกับข้อผิดพลาดในดาต้าแกรมก่อนหน้านี้

ถ้ารูทีนของ TLI เกิดข้อผิดพลาดสามารถที่จะตรวจสอบดูได้จากตัวแปร t_errno รูทีนของ TLI จะยอมให้จัดการติดต่อสื่อสารในเน็ตเวิร์กภายในแอพพลิเคชั่นจะต้องมีการกระทำดังนี้

1. การกระทำของแอพพลิเคชั่นเลเยอร์จะต้องกำหนดบริการต่างๆระบุข้อมูลและบริการที่ต้องการและรายละเอียดผลลัพธ์ของบริการและจะต้องอิมพลีเมนต์บริการต่างๆบนเครื่องเซิร์ฟเวอร์
2. การกระทำของพรีเซนต์เตชั่นเลเยอร์ จะต้องแทนที่ข้อมูลทั้งหมดในรูปแบบเดียวกัน
3. การกระทำจของเซสชั่นเลเยอร์ แอพพลิเคชั่นจะต้องจัดการเซสชั่นโดยการเตรียมการ “over” และ “over and out” ข้อความที่เกิดขึ้นในเวลาเดียวกัน

2.4 Windows Sockets

Windows Sockets เป็นตัวที่ใช้กำหนดการอินเทอร์เฟซ(Interface) การโปรแกรมบนเน็ตเวิร์กสำหรับไมโครซอฟท์วินโดว์ โดยเป็นมาตราฐานของ BSD ซึ่งประกอบไปด้วยชุดของรูทีนและกลุ่มของ Window-specific ที่เพิ่มเติมออกมาซึ่งมันจะช่วยให้โปรแกรมเมอร์ใช้ข้อดีของ message-driven ของวินโดว์ จุดมุ่งหมายของวินโดว์ซ็อคเก็ตคือเตรียมชุด API เพื่อช่วยใช้การพัฒนาโปรแกรมและให้ซอฟท์

แวร์แต่ละอย่างทำงานได้อย่างตรงกัน วินโดว์ซ็อคเก็ตทั้งหมดจะรองรับการทำงานทั้งแบบสตรีม (TCP) และแบบดาต้าแกรม (UDP)

API มีจุดมุ่งหมายที่จะให้สามารถใช้งานภายในวินโดว์เวอร์ชั่นต่างๆได้ ซึ่งจะทำให้วินโดว์ซ็อคเก็ตแอพพลิเคชั่นสามารถทำงานได้ทั้งสภาพแวดล้อมแบบ 16 และ 32 บิต และวินโดวีซ็อคเก็ตจะช่วยให้การพัฒนาซอฟท์แวร์ตรงกับรูปแบบของการโปรแกรมบนวินโดว์

2.4.1 Socket concept

พื้นฐานของการสร้างบล็อกสำหรับการติดต่อสื่อสารเรียกว่า ซ็อคเก็ต (Socket) ซึ่งเป็นจุดเอ็นต์พอยต์ (Endpoint) ของการติดต่อสื่อสาร ซ็อคเก็ตโดยปกติจะแลกเปลี่ยนข้อมูลกันภายในโดเมนเดียวกัน และการใช้งานแต่ละชนิดขึ้นอยู่กับแต่ละโปรเซส (Process) วินโดว์ซ็อคเก็ตจะช่วยให้การติดต่อสื่อสารมีความสะดวกสบายขึ้นทั้งแบบที่อยู่ในโดเมนเดียวกัน หรืออินเทอร์เนตโดเมน ซ็อคเก็ตในปัจจุบันมีอยู่สองชนิดคือ อย่างแรกเรียกว่า สตรีมซ็อคเก็ต (Stream Socket) เป็นลักษณะสองทิศทาง มีความน่าเชื่อถือในการส่งข้อมูล อีกชนิดคือ ดาต้าแกรมซ็อคเก็ต (Datagram Socket) มีลักษณะสองทิศทางเช่นกัน แต่ความน่าเชื่อถือในการส่งข้อมูลยังไม่น่าเชื่อถือเท่าไหร่นัก

2.4.2 Client-Server Model

เป็นรูปแบบโครงสร้างของแอพพลิเคชั่นกระจายการทำงาน ซึ่งฝั่งไคลเอ็นต์ (Client) จะทำการร้องขอบริการจากฝั่งเซิร์ฟเวอร์ (Server) จะเป็นการติดต่อสื่อสารระหว่างไคลเอ็นต์และเซิร์ฟเวอร์ โปรโตคอลที่ใช้ในการติดต่อมีอยู่สองรูปแบบคือ ในแบบแรกจะเป็นซิมเมตริกโปรโตคอล (Symmetric Protocol) ทั้งสองด้านจะทำหน้าที่คล้ายมาสเตอร์และสลาฟ แต่ในอีกแบบคืออซิมเมตริกโปรโตคอล (Asymmetric Protocol) การทำงานด้านหนึ่งจะอยู่นิ่งเป็นราวกับว่ามาสเตอร์ (Master) แต่ในอีกด้านหนึ่งจะเป็นสลาฟ (Slave) ตัวอย่างของซิมเมตริกโปรโตคอลจะเป็นพวกโปรโตคอลเทลเน็ต (Telnet Protocol) ที่ใช้เป็นการรีโมทล็อกอินจากที่ไกลๆ ส่วนตัวอย่างแบบอซิมเมตริกจะเป็นพวกโปรโตคอลไฟล์ทรานสเฟอร์ (File Transfer protocol)

2.4.3 Byte Ordering

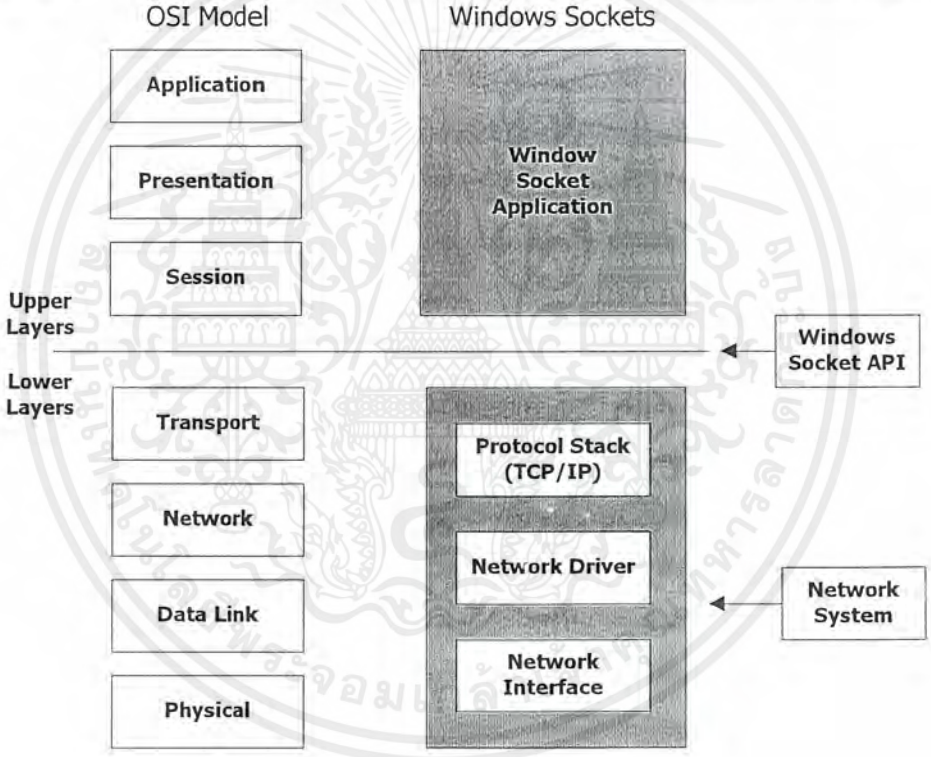
จะเป็นการอ้างอิงถึง IP Address หรือหมายเลขพอร์ตที่จะส่งหรือจากรูทีนของวินโดว์ซ็อคเก็ตที่จะต้องอยู่ในเน็ตเวิร์กออเดอร์ (Network order) ซึ่งรวมทั้งค่า IP Address และพอร์ตฟิลด์ของ Struct sockaddr_in ในการที่จะทำการติดต่อกับเซิร์ฟเวอร์จะต้องรู้หมายเลขพอร์ตซึ่งจะหาได้จากค่าที่จะส่งกลับมาของรูทีน getservbyname() และตัวแอพพลิเคชั่นต้องใช้รูทีนในการเปลี่ยนตัวเลขของโฮสต์ไปเป็นเน็ตเวิร์กคือ htons() ถ้าแอพพลิเคชั่นต้องการแสดงหมายเลขของพอร์ต หมายเลขของพอร์ตจะต้องถูกเปลี่ยนจากเน็ตเวิร์กไปเป็นโฮสต์โดยใช้รูทีน ntohs()

2.4.4 Winsock Network Model

ตามโครงสร้างด้านบนเมื่อเทียบกับ OSI Model จะเห็นได้ว่าด้าน Upper Layers ของ OSI Model จะเป็นเหมือนกับ Windows Socket Application ของ Winsock Model จากโครงสร้างของ Winsock Model จะประกอบไปด้วยส่วนต่างๆใหญ่ๆสามส่วนดังนี้

WinSock application	จัดการเกี่ยวกับฟังก์ชั่นการทำงานระดับบน
Network System	จัดการเกี่ยวกับฟังก์ชั่นการทำงานระดับต่ำ
WinSock API	ยอมให้ระดับบนและระดับล่างมีการติดต่อกันได้

WinSock Application เป็นการรันโดยตัวมันเดี่ยวๆมันอาจจะเป็น Dynamic Link Library (DLL) กับ API ระดับสูงและแอพพลิเคชั่นของตัวเอง อาจจะเป็นสแต็กของ DLLs กับแอพพลิเคชั่นด้านบน WinSock API(WSA) จะจัดการเกี่ยวกับเข้าถึงระบบเน็ตเวิร์ก และ Winsock Application จะใช้บริการนี้ในการรับส่งข่าวสารข้อมูล ซึ่ง WSA นี้จะเป็นตัวที่ใช้แยกระหว่างเลเยอร์ระดับบนและเลเยอร์ระดับล่าง



รูปที่ 2.4-1 การเปรียบเทียบระหว่าง OSI Model และ Winsock Model

ลักษณะการทำงานของ WinSocks

แอพพลิเคชั่นที่ใช้วินโดว์ซ็อคเก็ตจะทำงานที่ระบบวินโดว์ 3.1 ขึ้นไป จะต้องทำการติดตั้ง Winsock..DLL และตัว Networking Stack ทั้งสองตัวนี้จะสามารถเลือกใช้ได้ตามด้านการนำไปใช้งาน เมื่อแอพพลิเคชั่นเริ่มทำงานมันจะไปทำการหาไฟล์ Winsock.DLL โดยทำตามขั้นตอนนี้คือ

1. เริ่มหาจากไดเรกเทอรีปัจจุบัน

2. ต่อมาหาจากไดเรกเทอรีวินโดว์
3. ต่อมาหาจากไดเรกเทอรีซิสเต็มของวินโดว์
4. หาจากไดเรกเทอรีที่เป็นเส้นทางสภาพแวดล้อม
5. ต่อจากนั้นลองหาจากไดเรกเทอรีที่ทำการแมปไว้ในเน็ตเวิร์ก

ไฟล์ที่ใช้สำหรับการพัฒนา Windows Sockets Application

1. WINSOCK.H บรรจุข้อมูลทั้งหมดที่ใช้โดย Windows Sockets เป็นไฟล์มาตราฐานที่ใช้ในการพัฒนา Winsock Application ที่พัฒนาโดยภาษา C หรือ C++
2. WINSOCK.DEF เป็นไฟล์ที่ใช้บอกสิ่งต่างๆที่อยู่ในไฟล์ WINSOCK.DLL
3. WINSOCK.LIB เป็นไฟล์ที่เก็บไลบารี่ต่างๆที่ใช้งาน
4. Help file เป็นส่วนที่ใช้ช่วยเหลือในการเขียนโปรแกรมหรือพัฒนาโปรแกรม

2.4.5 Socket Library Overview

Socket function

- accept() เป็นการตอบรับการติดต่อที่ร้องขอเข้ามา
- bind() กำหนดชื่อเฉพาะให้ซ็อคเก็ตที่ไม่มีชื่อ
- closesocket() ปิดการเชื่อมต่อของซ็อคเก็ต
- connect() เริ่มต้นการติดต่อกับรีโมทแอดเซสที่ระบุ
- htonl() เปลี่ยนเลข 32 บิตของโฮสไปเป็นเน็ตเวิร์ก
- htons() เปลี่ยนเลข 16 บิตของโฮสไปเป็นเน็ตเวิร์ก
- ntohl() เปลี่ยนเลข 32 บิตของเน็ตเวิร์กไปเป็นของโฮส
- ntohs() เปลี่ยนเลข 16 บิตของเน็ตเวิร์กไปเป็นของโฮส
- getsockname() รับเอาชื่อปัจจุบันของซ็อคเก็ตที่ระบุ
- listen() คอยรับฟังการร้องขอที่เข้ามา
- rcv() รับข้อมูลจากซ็อคเก็ตที่ติดต่อเข้ามา
- rcvfrom() รับข้อมูลจากซ็อคเก็ตที่ติดต่อเข้ามาและไม่ได้ติดต่อเข้ามา
- send() ส่งข้อมูลไปยังซ็อคเก็ตที่ติดต่อเข้ามา
- sendto() ส่งข้อมูลไปยังซ็อคเก็ตที่ติดต่อเข้ามาและไม่ได้ติดต่อเข้ามา
- shutdown จบการติดต่อสื่อสาร
- socket() สร้างเอ็นดพอยต์ของการสื่อสารและส่งค่ากลับมาเป็นซ็อคเก็ต
- inet_addr() เปลี่ยนรูปแบบอักษรที่มีจุดไปเป็นค่า Internet address
- inet_ntoa() เปลี่ยนค่า Internet address ไปเป็นรูปแบบอักษรที่มีจุด

ฟังก์ชั่นเกี่ยวกับดาต้าเบส

- gethostbyaddr() รับเอาชื่อและแอดเดรสที่มีลักษณะตรงกับเน็ตเวิร์กแอดเดรส
- gethostbyname() รับเอาชื่อและแอดเดรสที่มีลักษณะตรงกับชื่อโฮสต์

- gethostname() รับเอาชื่อของโลคอลโฮสต์
- getprotobyname() รับเอาชื่อโปรโตคอลและจำนวนที่ตรงกับชื่อโปรโตคอล
- getprotobynumber() รับเอาชื่อโปรโตคอลและจำนวนที่ตรงกับหมายเลขโปรโตคอล
- getservbyname() รับเอาชื่อเซอร์วิสและพอร์ตที่ตรงกับชื่อเซอร์วิส
- getservbyport() รับเอาชื่อเซอร์วิสและพอร์ตที่ตรงกับพอร์ต

ฟังก์ชั่นที่ระบุมาจากการขยายของไมโครซอฟท์วินโดว์

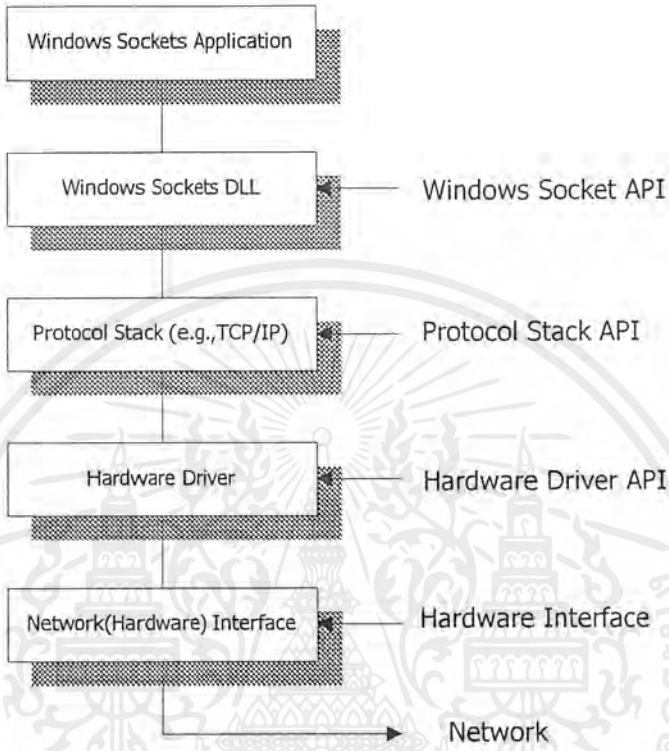
- WSAAsyncGetHostByAddr() เป็นกลุ่มของฟังก์ชันที่จัดการเกี่ยวกับอะซิงโครนัส
  WSAAsyncGetHostByName() ตัวอย่างเช่นฟังก์ชั่น WSAAsyncGetHostByName
  WSAAsyncGetProtoByName() เป็นฟังก์ชั่นเกี่ยวกับการจัดการข้อความแบบอะซิงโครนัส
  WSAAsyncGetServerByName() โดยอิมพลีเมนต์มาจากฟังก์ชั่น gethostbyname()
  WSAAsyncGetProtoByNumber()
  WSAAsyncGetServerByPort()
- WSAAsyncSelect() จัดการเกี่ยวกับอะซิงโครนัสเวอร์ชันของ Select()
- WSACancelAsyncRequest() ยกเลิกข้อมูลที่ค้างของฟังก์ชั่น WSAAsyncGetXByY
- WSACancelBlockingCall() ยกเลิกบล็อคที่ค้างของ API Call
- WSACleanup() เป็นตัวแสดงหยุดการทำงานของวินโดว์ซ็อคเก็ต DLL
- WSAGetLastError() บรรจุรายละเอียดข้อผิดพลาดต่างๆ
- WSAIsBlocking() ใช้เมื่อวินโดว์ซ็อคเก็ต DLL นั้นยังคงถูกใช้จากเธรด
- WSASetBlockingHook() "HOOK" เป็นวิธีการบล็อคกิ้งโดยวินโดว์ซ็อคเก็ต
- WSASetLastError() เซตข้อผิดพลาดที่ส่งกลับมาโดยซับซีเควน
- WSAStartup() ค่าเริ่มต้นของวินโดว์ซ็อคเก็ต DLL
- WSAUnhookBlockingHook() นำเอาฟังก์ชันบล็อคแรกกลับคืนมา

ความสำคัญระหว่าง Window Socket กับส่วนประกอบเน็ตเวิร์กอื่นๆ

Winsock เป็น (Application Programming Interface หรือ API) ที่ใช้ในการเข้าถึงฟังก์ชันเน็ตเวิร์ก WINSOCK.DLL สำหรับแอพพลิเคชั่นขนาด 16 บิต และ WSOCK32.DLL สำหรับแอพพลิเคชั่นขนาด 32 บิต WINSOCK.DLL เป็นการพัฒนาของ Windows Sockets specification ที่เตรียมการเกี่ยวกับเน็ตเวิร์ก โดยการเข้าถึงเน็ตเวิร์กโปรโตคอลสแต็ก หลักการหรือการอินเทอร์เฟซ WINSOCK.DLL จะอิมพลีเมนต์โดยใช้การติดต่อสื่อสารกับเน็ตเวิร์กสแต็ก เป็นหลักการคุณสมบัติที่ระบุโดยผู้ขาย

วินโดว์ซ็อคเก็ตจะทำการกำหนดอินเทอร์เฟซสำหรับที่จะเข้าถึงเน็ตเวิร์กโปรโตคอล แต่มันจะไม่ใช่โปรโตคอลของมันเอง ดังนั้นวินซ็อคแอพพลิเคชันอาจจะติดต่อสื่อสารไม่ใช่กับวินซ็อคแอพพลิเคชั่น อย่างอื่น

อย่างเดียว แต่จะต้องติดต่อกับแอพพลิเคชันที่ใช้เน็ตเวิร์กโปรโตคอลสแต็กอย่างเดียวกันเช่น แอพพลิเคชันของยูนิกซ์ที่ใช้ซ็อคเก็ต



รูปที่ 2.4-2 ความสัมพันธ์ระหว่าง Application, WINSOCK.DLL และส่วนต่างๆของเน็ตเวิร์ก

2.5 Remote Procedure Calls

Remote Procedure Call (RPC) เป็นเทคนิคในการสร้างระบบแบบกระจาย โดยยอมให้โปรแกรมบนเครื่องหนึ่งเรียกซับรูทีนบนเครื่องอื่นได้ ในแอพพลิเคชั่นทั่วๆไปจะมีการเรียก Procedure Call เมื่อเทียบกับ Client/Server Model แล้ว Main Program ก็เปรียบได้กับไคลเอนต์โดยผ่านค่าไปยังโพรซีเดอร์ที่เป็นเซิร์ฟเวอร์ใน RPC จะทำงานในลักษณะโปรเซสที่ทำบน Local System เรียกใช้โพรซีเดอร์ที่อยู่บน Remote System

2.5.1 โครงสร้างของ remote procedure call

โครงสร้าง rpc ประกอบด้วย 3 ส่วน คือ แอพพลิเคชันบนฝั่งไคลเอนต์ , โพรซีเดอร์บนฝั่งเซิร์ฟเวอร์ และโปรแกรมที่แก้ปัญหา แอดเดรสซึ่งเรียกว่า rpcbind (บาง unix system จะเรียกว่า port map)

rpcbind program เป็น deamon program ที่รันบนเครื่องเซิร์ฟเวอร์โดยจะเก็บตารางที่ map rpc procedure ที่ใช้ทั้งหมดลงบน transport address เมื่อแอพพลิเคชั่นบนเครื่องไคลเอนต์ต้องการเรียก

Remote procedure เริ่มแรกจะส่ง Query ไปที่ rpcbind process บนเครื่องเซิร์ฟเวอร์เพื่อ get transport address ของโพรซีเดอร์

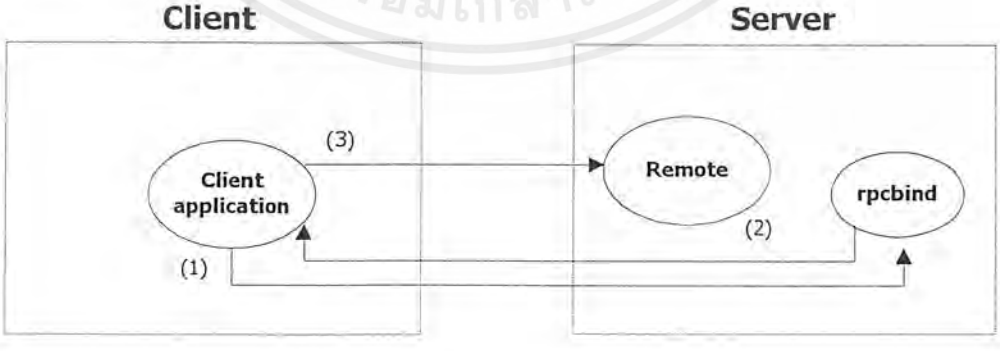
แอพพลิเคชั่นบนเครื่องไคลเอนต์ต้องรู้ Transport address ของ rpcbind process บนเครื่องเซิร์ฟเวอร์ดังนั้น rpcbind process ต้องหา Well-know transport address ให้แอพพลิเคชั่นที่เรียกนั้นทำงาน ดังนี้ เมื่อ rpcbind process เริ่มทำงาน มันจะหา transport provider บนระบบโดยใช้ Network Selection routine ซึ่งในแต่ละ transport provider จะทำการติดต่อกับ well-know address นั่นคือ IP address ของเครื่องเซิร์ฟเวอร์ที่ถูกรวมกับ Port 111 เมื่อใช้ well-know address บน TCP/IP

โพรซีเดอร์บนเครื่องเซิร์ฟเวอร์ในการที่จะทำให้โพรซีเดอร์สามารถทำงานบนเครือข่ายได้นั้นจะต้องเขียนโพรซีเดอร์และใช้ RPC specification language ในการ encapsulate เข้าไปในโปรแกรมซึ่งโปรแกรมนี้จะรันเป็น deamon บนฝั่งเครื่องเซิร์ฟเวอร์โดยมีการทำงานดังนี้

1. กำหนด transport provider บนระบบ
2. ติดต่อแอดเดรสที่ไม่ได้ถูกใช้ในแต่ละ transport provider ด้วยตัวเอง
3. ส่งแมสเซสไปที่ rpcbind deamon ผ่าน local loopback และจะได้แอดเดรสที่ต้องการ

โดยจะมี 2 โพรซีเดอร์กำลังรันอยู่ตลอดเวลาคือ rpcbind และโปรแกรมที่เก็บโพรซีเดอร์ดังนั้น แอพพลิเคชั่นบนฝั่งเครื่องไคลเอนต์สามารถเรียก remote procedure ได้จากรูปที่ 2.5.1-1 แสดงการทำงานของ RPC ในการที่ไคลเอนต์สามารถติดต่อไปยังเซิร์ฟเวอร์ได้มีลำดับขั้นตอนดังรูปที่ 2.5.1-1 ดังนี้

1. เมื่อ RPC server(deamon) เริ่มทำงานมันจะได้รับแอดเดรส (port number) เพื่อรอรับ request มา ซึ่งจะทำการ register address ไว้กับ rpcbind หรือ portmapper นอกจากนั้นยังต้องทำการ register program number , program version ไว้ด้วย
2. ก่อนที่ไคลเอนต์จะติดต่อกับเซิร์ฟเวอร์ได้ rpcbind จะทำการกำหนดพอร์ต (address) ที่จะคอยรับฟัง request ไว้ก่อน
3. หลังจากนั้นไคลเอนต์จะสามารถติดต่อกับเซิร์ฟเวอร์ผ่านทาง path ที่กำหนดไว้โดยไคลเอนต์จะ request ไป ส่วนเซิร์ฟเวอร์ก็จะทำการ reply กลับมา



รูปที่ 2.5.1-1 RPC setup

2.5.2 การสร้าง RPC Program

ในการสร้าง RPC Program จะต้องสร้างไฟล์ทั้งหมด 3 ไฟล์ดังนี้ ไฟล์ ' .x' สำหรับเก็บ RPC Specification ที่เขียนใน RPC definition language ไฟล์ ' .c' ซึ่งสร้างในฝั่งเซิร์ฟเวอร์สำหรับ remote procedure และไฟล์ ' .c' ในฝั่งไคลเอนต์ที่เก็บแอพพลิเคชั่นที่เรียก remote procedure ซึ่งมีขั้นตอนดังนี้

1. กำหนด set และ group ของโพรซีเดอร์ โดยการระบุหมายเลขของโปรแกรมและหมายเลขเวอร์ชัน
2. กำหนดแต่ละโพรซีเดอร์ว่าทำอะไรบ้าง โดยการระบุอาร์กิวเมนต์ไปที่โพรซีเดอร์และกำหนดว่าโพรซีเดอร์จะส่งค่าอะไรกลับมา
3. อิมพลีเมนต์โพรซีเดอร์ทั้งหมด
4. สร้างแอพพลิเคชั่นฝั่งไคลเอนต์ที่จะเรียก remote procedure

SVR4 ได้เตรียมคอมไพเลอร์ที่ช่วยในการเขียนโปรแกรมโดยคอมไพเลอร์จะนำ high-level specification ของ remote procedure และ กำเนิด C code ที่ใช้เรียก lower-level RPC routine

2.5.3 การใช้ RPC compiler

ในการสร้างแอพพลิเคชั่นที่ใช้กลไกของ RPC นั้นจะต้องใช้ lower-level RPC routine ในการ handle network และเขียน XDR routine เพื่อแปลงข้อมูลให้อยู่ในรูปแบบที่เข้าใจตรงกัน โดย SVR4 ได้เตรียมคำสั่ง rpcgen เพื่อช่วยในการสร้างโปรแกรม RPC rpcgen เป็นคอมไพเลอร์บนระบบปฏิบัติการยูนิกซ์ที่สนับสนุนการพัฒนาโปรแกรม RPC การพัฒนาโปรแกรมด้วย rpcgen นั้น ไฟล์ที่จะเป็นอินพุทให้กับ rpcgen นั้นจะเป็นเท็กซ์ไฟล์ที่ประกอบด้วยส่วนต่างๆดังนี้

- หมายเลขโปรแกรม RPC (An RPC program number)
- หมายเลขเวอร์ชันของโปรแกรม RPC หนึ่งหมายเลขหรือมากกว่า (RPC program version number)
- หมายเลขของโพรซีเดอร์หรือฟังก์ชัน RPC หนึ่งหมายเลขหรือมากกว่า
- ดาต้าไทป์ที่จะผ่านค่าไปยัง RPC ฟังก์ชัน (rpcgen จะสร้างเป็น XDR ให้โดยอัตโนมัติ)
- ออปชันของภาษาซีที่จะทำการคัดลอกลงไปในไฟล์เอาท์พุตที่จะได้จาก rpcgen

RPC ฟังก์ชันจะถูกระบุโดยหมายเลขโปรแกรม RPC , หมายเลขเวอร์ชันของโปรแกรม RPC และหมายเลขของโพรซีเดอร์หรือฟังก์ชัน RPC

RPC โปรแกรมหนึ่งโปรแกรมจะไปเป็นโปรเซสหนึ่งโปรเซสของเซิร์ฟเวอร์ ซึ่งโปรเซสนี้จะมีการตอบสนองหรือมีการทำงานกับโพซีเดอร์ที่ได้ระบุไว้ในโปรแกรมที่ส่วนของไคลเอนต์

หมายเลขเวอร์ชัน RPC จะเป็นตัวระบุลำดับการแก้ไขหรือปรับปรุงกลุ่มฟังก์ชันของ RPC โดยจะเป็นตัวเลขจำนวนเต็มและควรจะเริ่มด้วยหมายเลข 1

หมายเลขของโพรซีเดอร์หรือฟังก์ชัน RPC จะเป็นหมายเลขที่กำหนดให้กับโพรซีเดอร์หรือฟังก์ชันที่ไม่ซ้ำกัน ถ้ามีหมายเลขที่ซ้ำกัน โปรแกรมจะถือเอาตัวที่มีการเปลี่ยนแปลงของหมายเลขโปรแกรม RPC หรือ หมายเลขของโพรซีเดอร์หรือฟังก์ชัน RPC เท่านั้น

อินพุทไฟล์จะถูกเขียนใน RPC definition language ซึ่งจะเก็บอยู่ในรูปของไฟล์ ' .x ' โดยไฟล์นี้จะเก็บข้อกำหนดของโครงสร้างข้อมูลที่ใช้ในแต่ละ โพรซีเดอร์ เมื่อนำไฟล์นี้ไปคอมไพล์ด้วย rpcgen จะได้ไฟล์ C-type ต่างๆและไฟล์ header ดังนี้

file.h เป็น header ไฟล์

file_svc.c เก็บ C code ที่เป็นฟังก์ชั่น rpc ระดับต่ำ โดยไฟล์นี้จะมี code ที่ register program ด้วย rpcbind deamon

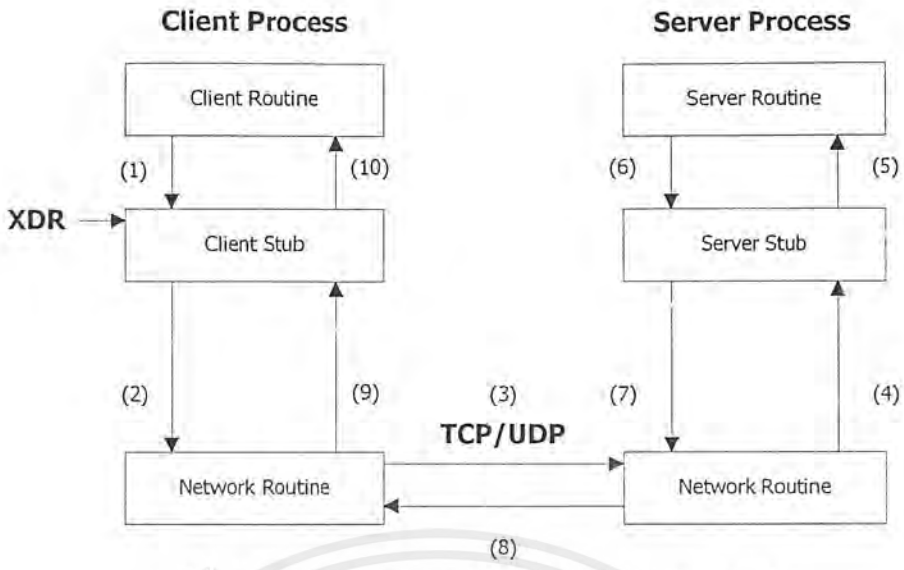
file_clnt.c เก็บ C code ที่ implement ฟังก์ชั่น RPC Client ในระดับต่ำ

file_xdr.c เก็บ XDR routine ทั้งหมดที่แปลง Data ให้อยู่ในรูป XDR format

2.5.4 การทำงานของ Remote Procedure Call

RPC ใช้โมเดลแบบ Request and Reply Model โดยโพรซีเดอร์ฝั่งไคลเอนต์จะส่ง Request message ไปยังโพรซีเดอร์ฝั่งเซิร์ฟเวอร์ซึ่งจะส่ง Reply messages กลับมา โดยการติดต่อทั้งสองฝั่งจะผ่านตัวจัดการคือ client stub และ server stub โดยตัว stub เป็น communication interface ซึ่งจะเป็นเครื่องมือสร้าง RPC Protocol และข้อกำหนดของการสร้างและแลกเปลี่ยนแมสเซสโดยตัว Stub จะเก็บ ฟังก์ชั่น ที่ map local procedure calls ให้อยู่ในรูปแบบ network rpc function calls โดย Client stub จะใช้ library (RPC library) เป็นตัวจัดการการติดต่อระหว่าง 2 process) เป็นตัวช่วยหา remote process และฝั่ง server stub จะจัดการในระดับล่างของฟังก์ชั่น

ในการติดต่อระหว่างไคลเอนต์และเซิร์ฟเวอร์จะต้องใช้รูปแบบที่เข้าใจคือ Machine Independent Data Representation แต่ในการติดต่อระหว่างเครือข่ายนั้นจำเป็นต้องแปลงให้อยู่ในรูปแบบที่กำหนดคือ External Data Representation(XDR) โดยตัวที่ทำหน้าที่ในการแปลงรูปแบบข้อมูลทั้งสองแบบของฝั่งไคลเอนต์และเซิร์ฟเวอร์คือ Client Stub และ Server Stub (โดย XDR routine จะเก็บตัว Filter ที่ใช้สำหรับแปลให้เป็น C-type) จากรูปที่ 2.5.4-1 การทำงานของ Remote Procedure Call เป็นลำดับได้ดังนี้



รูปที่ 2.5.4-1 การทำงานของ Remote Procedure Call(RPC)

1. โพรซีเดอร์บนไคลเอนต์จะทำการเรียก Client Stub โดยจะส่งออกไปในรูปมาตรฐานในรูปแบบ Marshaling

2. ข้อมูลที่จะส่งไปยัง Remote System โดย client Stub ต้องเรียกผ่าน System Cal บน Local Kernel

3. ข้อมูลจะถูกส่งผ่านใน 2 ลักษณะคือแบบ Connection-Oriented และแบบ Connectionless

4. ส่วนบน Server Stub จะทำงานเหมือน Local Procedure แต่กระทำบนเซิร์ฟเวอร์และนำผลที่ได้ผ่าน ไปยัง Client Stub โดยผ่านเน็ตเวิร์กต่อไป

5. Server Stub จะทำงานเหมือน Local Procedure แต่กระทำบนเซิร์ฟเวอร์และนำผลที่ได้ผ่านไปยัง Client Stub โดยผ่านเน็ตเวิร์กต่อไป

6. เมื่อโพรซีเดอร์บน Server Stub ทำงานจบก็ส่งค่าต่างๆไปยัง Server Stub

7. Server Stub จะทำการแปลงค่าต่างๆที่ต้องการส่งกลับและ Marshal ส่งกลับไปยัง Client Stub

8. ข้อมูลจะถูกส่งผ่านเน็ตเวิร์กไปยัง Client stub

9. Client Stub จะอ่านข้อมูลที่ได้มาจากเน็ตเวิร์กโดยผ่าน Local Kernel

10. หลังจากข้อมูลได้ผ่าน Client Stub เรียบร้อยแล้วค่าที่ได้จะถูกส่งไปยังโพรซีเดอร์ฝั่งไคลเอนต์ที่เรียกใช้ Remote system ต่อไป

สาระของ RPC คือการซ่อน Network code ทั้งหมดไว้ใน Stub procedure ในแอพพลิเคชั่นบนไคลเอนต์และเซิร์ฟเวอร์ ไม่ต้องมีขั้นตอนที่ยุ่งยากเหมือนซ็อกเก็ตหรือแบบอื่นๆ RPC จะช่วยในการเขียน Distributed Application เป็นเรื่องที่ง่ายขึ้น ถ้าเรานำ RPC ไปเปรียบเทียบกับ OSI Model แล้วจะอยู่ระหว่าง Transport Layer ภายในส่วนของ Spoliation Layer RPC จะเข้าไปจัดการเกี่ยวกับเรื่องการติดต่อสื่อสาร(Networking Detail) โดยจะนำส่วนที่ต้องการส่งออกไปนำไปไว้ในอาร์กิวเมนต์เพื่อแปลงให้อยู่ในรูปแบบมาตรฐานเพื่อใช้ในการสื่อสารระหว่างไคลเอนต์กับเซิร์ฟเวอร์ โดยที่ Application Layer ไม่ต้องสนใจเรื่อง Byte Order

2.5.5 External Data Representation(XDR)

ข้อมูลบน Remote machine อาจมีรูปแบบไม่เหมือนกับ Local machine ถ้าแอพพลิเคชั่นบนเครื่องไคลเอนต์ส่งข้อมูลไปที่โพรซีเดอร์บนเครื่องเซิร์ฟเวอร์ความแตกต่างของรูปแบบอาจเป็นเหตุให้ remote procedure แปลข้อมูลผิดไปได้ สำหรับการแก้ปัญหานี้คือข้อมูลทั้งหมดที่ผ่านเข้าไปใน remote procedure และ return กลับมาอาจจะเข้ารหัสในรูปแบบเดียวกันเพื่อให้เข้าใจตรงกันโดย SVR4 เตรียม package ที่เรียกว่า XDR สำหรับการทำงานนี้ซึ่ง XDR จะรวมเป็นกลไกหนึ่งของ RPC

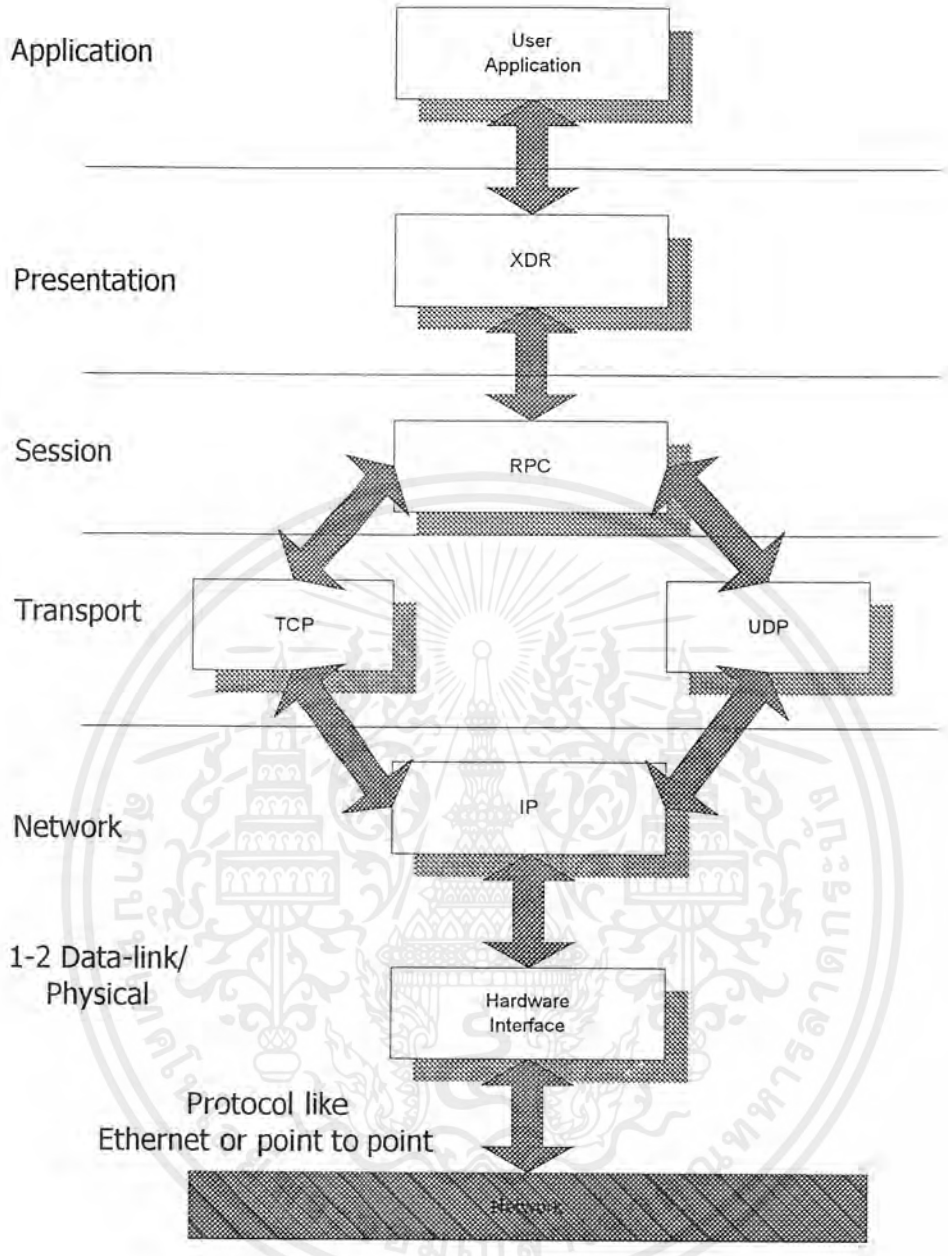
Overview ของ XDR

1. Data Size ขนาดของข้อมูล
2. Byte Ordering การเสนอข้อมูล เช่น most significant byte ซึ่งบางเครื่องอาจอยู่ใน right byte
3. Data Representation ชนิดข้อมูลอาจแตกต่างกัน เช่น บางเครื่อง string อาจปิดด้วย null บางเครื่องอาจปิดด้วยความยาว
4. Alignment บางเครื่องอาจมีข้อจำกัดบนขอบเขตของ address

แอพพลิเคชั่นสามารถใช้ XDR routine ในการแปลงจาก local representation data ไปเป็น XDR Representation data ก่อนที่จะส่งไปในเครือข่าย และที่ฝั่ง remote machine จะใช้ XDR routine ในการแปลงจาก XDR representation data ไปเป็น remote representation data หลังเมื่อรับข้อมูลแล้ว ดังนั้นสามารถใช้ XDR ในการ representation ชนิดของข้อมูลใดๆก็ได้ หลักสำคัญคือ ผู้ส่งและผู้รับต้องรู้รูปแบบของข้อมูลขณะส่ง

2.5.6 การเปรียบเทียบระหว่าง RPC และ OSI Model

จาก OSI ที่มี 7 layer สามารถอธิบาย RPC ว่าทำงานในระดับใดบ้างดังรูป 2.5.6-1 โดย โปรแกรมจะอยู่ใน Top layer ONC-RPC 4.0 Library จะถูก support โดย TCP, UDP และ IP protocol



รูปที่ 2.5.6-1 ตำแหน่งของ XDR และ RPC ที่อยู่ภายในมาตราฐาน OSI

2.6 การพัฒนาแอพพลิเคชั่นโดยใช้ RPC

ก่อนที่จะเขียน RPC แอพพลิเคชั่น อาจจะมีการใช้คำสั่ง shell ของยูนิกซ์มาช่วยในการพัฒนาโปรแกรม ซึ่งมีปัญหาหลักอยู่ 2 ปัญหาคือ คำสั่งทำงานช้าและจะต้องมี login บน remote machine ในกรณีที่จะใช้คำสั่ง shell ติดต่อกับโปรแกรมบน remote machine ทางแก้คือให้ออกแบบสร้างโปรแกรมที่ฝั่งเซิร์ฟเวอร์ที่สามารถ request และ reply ได้อย่างรวดเร็วนั่นเอง

การพัฒนา RPC แอพพลิเคชั่นมีอยู่ 2 ขั้นตอนคือ

1. กำหนดโปรโตคอลสำหรับการติดต่อ Client และ Server

2. พัฒนาโปรแกรมไคลเอนต์และเซิร์ฟเวอร์

หลังจากนั้นจะนำโปรแกรมมาคอมไพล์และลิงค์กับ stub และ library แล้วทำการทดสอบแอฟพิเคชั่น เมื่อได้แล้วจะนำโปรแกรมฝั่งเซิร์ฟเวอร์รันเป็น deamon และรันไคลเอนต์โปรแกรมที่โลคอล

2.6.1 การกำหนดโปรโตคอล

ในขั้นตอนแรกคือกำหนดการติดต่อระหว่างไคลเอนต์และเซิร์ฟเวอร์ โดยถ้าเราใช้ Protocol Compiler (RPCGEN) มันจะระบุชื่อของ Service procedure , data type ของ parameter และ return arguments ให้อัตโนมัติ โดยเราจะเขียน RPC language (RPCL) ซึ่งโครงสร้างภาษาจะคล้าย C language โดยมี extension เป็น .x

Example 1-1 RPCL protocol definition : rdb.x

```
/* preprocessor directives */
%#define DATABASE "personnel.dat"  /*  '%' passes it  through  */
/* constant definitions */
const  MAX_STR = 256;

/*structure definitions , no enumerations needed */
struct record {
        string firstName<MAX_STR>;     /* <> defines the maximum */
        string middleInitial<MAX_STR>;      /* possible length  */
        string lastName<MAX_STR>;
        int phone
        string location<MAX_STR>;
};
/* program definition , no union or type definitions needed */
program RDBPROG  {  /* could manage multiple servers  */
        version RDBVERS {
                record FIRSTNAME_KEY(string) = 1;
                record LASSTNAME_KEY(string) = 2;
                record PHONE_KEY(int) = 3;
                record LOCATION_KEY(string) = 4;
                int ADD_RECORD(record) = 5;
        } = 1;     /* you set the interface version number */
} = 0x20000001;  /*  program number ranges estabished by ONC  */
```

โดยสามารถนำ code.x มา compile ดังนี้

>rpcgen rdb.x /* rpcgen เป็น protocol compiler */

ตัว rpcgen จะสร้าง client stub (rdb_clnt.c) และ server stub (rdb_svc.c) และไฟล์ต่างๆที่จำเป็นต่อ XDR filter (เช่น rdb_xdr.c) และ Header file (rdb.h) ที่ใช้ร่วมกับ Application ของ client - server stub

2.6.2 การใช้งาน RPCGEN

RPCGEN จะมีอยู่ใน ONC RPC Release 4 สามารถเรียกใช้โดย command line ได้หลายแบบ ในแบบ แรกคือ

rpcgen *application.x*

ผลลัพธ์ที่ได้คือ header file, XDR, client และ server stub นอกจากนี้ยังสามารถใส่พารามิเตอร์เมื่อเรียกใช้ RPCGEN ได้ดังนี้

-D*name*[=*value*] เหมือนกับ #define

-I สร้าง code เพื่อให้สามารถใช้งานกับ inetd ได้

-K *seconds* เซิร์ฟเวอร์จะเลิกทำงานหลังจากที่ไม่มีกิจกรรมใดเกิดขึ้นภายในวินาทีที่กำหนด

-L เมื่อเกิดข้อผิดพลาดขึ้นจะเก็บค่าข้อผิดพลาดนั้นไว้ใน system logfile

-T สร้าง code เพื่อใช้งานกับ RPC dispatch table

-s *transport* สร้าง server code เพื่อให้สามารถใช้งานได้ใน transport protocol

-o *outfile* ชื่อของ output file

-c สร้าง xdr routines

-h สร้าง header file

-l สร้าง client stub

-m สร้าง server stub

-t สร้าง RPC dispatch table

RPCGEN New ที่อยู่ใน TIRPC จะมีพารามิเตอร์เพิ่มขึ้นดังนี้

-a สร้างทุกไฟล์รวมทั้งตัวอย่างของโปรแกรม client และ server

-b สามารถใช้ร่วมกับ RPC version เก่าได้

-C สร้างไฟล์ให้อยู่ในรูปแบบของ ANSI C

-i size

-N สนับสนุนการใช้อาร์กิวเมนต์หลายตัวและ call by value

-s nettype สร้าง server code ที่จะสนับสนุนโปรเซสของ DNS

-Sc	สร้าง client code ซึ่งใช้ remote procedure
-Ss	สร้าง server code ที่กำหนด remote procedure
-Y path	กำหนดไดเรกเทอรีที่เก็บ c preprocessor (cpp)

2.6.3 การกำหนดรูปแบบของภาษา (RPC Definition Language)

RPCL เป็น protocol descriptive language นำมาใช้ใน ONC RPCGEN ซึ่งเหมือนกับ XDR definition language แต่มีบางส่วนเพิ่มเติมขึ้นมา จะใช้เขียนใน *application.x* protocol definition ไฟล์ ลักษณะการเขียนนั้นจะคล้ายกับ C language

Definition

ใน RPCL protocol specification จะมีชนิดของ definition ทั้งหมด 6 ชนิด โดยในการเขียนนั้นไม่จำเป็นต้องมี definition ทุกชนิด และ ไม่จำเป็นต้องเขียนเรียงกันใน protocol specification จะเขียน definition ชนิดใดก่อนหลังก็ได้

Symbolic Constants

จะใช้เมื่อต้องการใช้อินทีเจอร์ในแบบค่าคงที่เช่น หากต้องการให้ค่า MAX_SIZE เท่ากับ 8192 ก็เขียนว่า

const MAX_SIZE = 8192;

หลังจาก compile แล้ว RPCGEN จะทำการแปลงเป็นคำสั่งของ C และปรากฏอยู่ใน *application.h* เป็นดังนี้

#define MAX_SIZE 8192;

ซึ่งทั้งไคลเอนต์และเซิร์ฟเวอร์จะนำไปใช้ได้

Enumeration

ใน RPCL นี้จะเหมือนกับใน C เช่น หากต้องการให้ define บาง color ไว้ก็ทำโดย

```
enum color {
        RED =0,
        GREEN = 1,
        BLUE = 2,
};
```

หลังจากใช้ RPCGEN แล้วจะเปลี่ยน code ไปอยู่ใน *application.h* เป็น

```
enum color {
        RED =0,
```

```
        GREEN = 1,
        BLUE = 2,
};
typedef enum color color;
bool_t xdr_color( );
```

ข้อแตกต่างโดยการใช้ RPCGEN ก็คือ RPCGEN จะใช้ typedef กำหนดให้ผู้เขียนโปรแกรมสามารถใช้ color ในการกำหนด type ได้เลย ไม่ต้องเขียน enum color เมื่อต้องการใช้ นอกจากนี้ RPCGEN จะกำหนด XDR encode/decode filter xdr_color( ) ใน *application_xdr.c* สำหรับใช้โดยทั้งไคลเอนต์ และเซิร์ฟเวอร์

Structures

ในการใช้งาน structure จะเหมือนกับใน C คือ

```
struct point {
        int x;
        int y;
};
```

ใน header file จะเปลี่ยนเป็น

```
struct point {
        int x;
        int y;
};
typedef struct point point;
bool_t xdr_point( );
```

ซึ่งจะทำให้เมื่อต้องการกำหนด type point ก็ใช้ point ได้เลยไม่ต้องใช้ struct point และนอกจากนั้นจะมีการกำหนด xdr_point ( ) ไว้ใน *application_xdr.c* ด้วย

Unions

XDR union จะคล้ายกับ variant record ใน pascal แต่จะไม่เหมือนกับ union ใน C เลย XDR union จะทำให้ง่ายในการ return error หรือ result โดยใช้เงื่อนไขในการตรวจสอบ เช่น หากไม่เกิด error ขึ้นบน server ก็จะ return data กลับมา แต่หากเกิด error ขึ้นก็จะไม่ส่งอะไรกลับมา

```
union result switch (int error) {
case 0:
        opaque data [MAX_SIZE];
```

```
default:
        void;
};
```

จาก code จะเห็นว่า error จะเป็นตัวใช้ในการตรวจสอบว่าควรจะส่ง data กลับไปหรือไม่ RPCGEN จะทำการ compile ไปไว้ใน *application.h* ว่า

```
union result switch (int error) {
case 0:
        opaque data [MAX_SIZE];
default:
        void;
};
typedef struct result result;
bool_t xdr_result ( );
```

จะเห็นว่าคล้ายกับ type ก่อนหน้านี้ คือสามารถกำหนด type โดยสั่ง result ได้เลย และจะมี xdr_result ( ) ซึ่งจะได้จากการ compile เป็น XDR filter ด้วย

Typedefs

โปรแกรมเมอร์สามารถกำหนด typedef ได้โดยตรง โดย RPCGEN จะไม่เปลี่ยนแปลง code ใดๆ เลย แล้วจะนำไปไว้ใน header file ให้ เช่น

```
typedef point poly [4]
```

Programs

เป็นส่วนเดียวที่แตกต่างกันระหว่าง RPCL กับ XDR ใช้สำหรับการกำหนด program name, program number, version และ procedure information ของ server หลังจาก compile ลงไปใน header file แล้ว client และ server stub จะนำไปใช้อีกทีหนึ่ง เช่น

```
program BOGUS_PROGRAM {
        version BOGUS_VERSION {
                void BOGUS_PROCEDURE (void) = 1;
        } = 1;
} = 0x2000000;
```

หลังจาก compile แล้วจะไปเก็บไว้ใน header file โดยเปลี่ยนเป็น

```
#define BOGUS_PROGRAM ((u_long) 0x2000000)
#define BOGUS_VERSION ((u_long) 1)
```

#define BOGUS_PROCEDURE ((u_long) 1)

extern void *bogus_procedure_1 ( );

ในไฟล์ definition หนึ่งๆนั้นจะสามารถ define server program ไว้ได้หลายโปรแกรมแต่ๆละโปรแกรม นั้นก็ต้องมีการกำหนดเวอร์ชั่นและโพรซีเดอร์ ไว้สำหรับแต่ละโปรแกรมหรือจะใส่ไว้หลายๆ version ด้วยก็ได้ จากตัวอย่างจะเห็นว่า โปรแกรมชื่อว่า BOGUS เป็น version 1 มีโพรซีเดอร์ ชื่อว่า BOGUS_PROCEDURE และ procedure number เป็น 1 โดยมีการกำหนด RPC program number ไว้เป็น 0x2000000 หลังจาก compile จะเห็นว่า มีการกำหนด bogus_procedure_1( ) ไว้ใช้เพื่อสะดวกในการพิมพ์เป็นตัวพิมพ์เล็กและบอกว่า procedure number เป็น 1 ด้วย

ใน ONC RPC นั้นจะมี procedure number เป็น 0 เรียกว่า NULLPROC อยู่เพื่อใช้ประโยชน์ในการ ping เซิร์ฟเวอร์เพื่อตรวจสอบว่าขณะนั้นเซิร์ฟเวอร์ยังทำงานดีอยู่ แต่หากใช้ RPCGEN นั้นไม่จำเป็นต้องกำหนดให้ procedure number เป็น 0 เพราะจะมีอยู่ใน service procedure อยู่แล้ว

Declarations

มีการ declare ทั้งหมด 4 ชนิดที่ใช้ใน RPCL

declaration:

simple-declaration

fixed-array-declaration

variable-array-declaration

pointer-declaration

simple-declaration จะเป็นเหมือนกับ C เช่น

color c;

จะแสดงเหมือนเช่นนี้ใน header file

fixed-length array

จะเหมือนกับใน C และแสดงเหมือนกันใน header file แต่ใน RPCL จะไม่สนับสนุน multi-dimention array เพราะในการส่งข้อมูลนั้นจะสามารถส่งได้เพียง offset ของข้อมูล เป็นแบบ one-dimention vector เท่านั้น หากต้องการใช้ก็อาจต้องดัดแปลงไปใช้ linked list หรือ tree แทน

color palette[8];

variable-length array

นั้นไม่มีใน C ใน XDR จึงใช้เครื่องหมาย angle bracket โดยใส่ขนาดของอาร์เรย์ไว้ระหว่างกลาง หากไม่ใส่ขนาดอาร์เรย์นั้นจะมีขนาดเท่าไรก็ได้

int x<MAX_SIZE>;

int y<>;

เนื่องจากในภาษา C ไม่มีจึงเปลี่ยนโครงสร้างไปเป็น struct ใน header file ดังนี้

```
struct {
        u_int x_len;
        int *x_val;
} x;
```

โดยขนาดของอาร์เรย์จะอยู่ใน _len และค่าที่เก็บไว้ในอาร์เรย์จะใช้ pointer _val ชี้ไป

pointer declaration

ใน XDR จะเหมือนกับใน C แต่ในการใช้งานนั้นไม่ควรจะมีการส่งแอดเดรสข้ามเน็ตเวิร์กควรจะใช้ในการสร้าง linked-list หรือ tree มากกว่า

```
pt *pNext;
```

Special Cases

Booleans

ในภาษา C นั้นจะไม่มี boolean type แต่ใน XDR และ RPCL libraries นั้นจะมี boolean_t ใช้สำหรับเก็บค่า TRUE หรือ FALSE เมื่อใช้ bool ใน RPCL แล้วจะเปลี่ยนเป็น bool_t ให้ใน header file

Strings

เนื่องจากในภาษา C ก็ไม่มีเหมือนกัน จึงใช้ NULL-terminated sequence มาช่วย คือเมื่อต้องการกำหนดเป็น string ก็ให้ใส่ขนาดไว้ใน angle bracket ไว้ โดยขนาดที่ใส่นี้ไม่รวม NULL หากไม่ใส่ขนาดไว้โปรแกรมจะกะขนาดให้เอง ขนาดของ string นี้จะมีผลกระทบกับ XDR encode/decode

```
string buffer<32>;
string longBuff< >;
```

หลังจาก compile จะได้ใน header file ว่า

```
char *buffer;
char *longBuff;
```

Opaque Data

XDR และ RPCL จะใช้ opaque data ในการเก็บตัวอักษร มีทั้งแบบ fixed size และแบบ variable size

```
opaque fixData [512];
opaque varData<1024>;
```

หลังจาก compile จะได้เป็น

```
char fixData [512];
struct {
        u_int varData_len;
        char *varData_val;
} varData;
```

Void

Void จะใช้เฉพาะใน union และ program เท่านั้น การเรียกใช้ void นั้นไม่ได้ใช้สำหรับต้องกำหนดตัวแปลขึ้นมา และโปรแกรมจะไปเรียก xdr_void( ) ให้ทำงาน

2.6.4 ฟังก์ชั่นของ ONC RPC ในระดับล่าง

หลังจากใช้ RPCGEN แล้วนั้น โปรแกรมเมอร์ยังจำเป็นต้องเขียน *application.c* ขึ้นมาเองอีกเพื่อทำงานฝั่งไคลเอนต์โดย RPCGEN จะสร้าง client stub ซึ่งทำงานเกี่ยวกับเน็ตเวิร์กขึ้นมาให้แล้ว แม้บางครั้งจะสามารถลดขนาดของ stub ลง หรือจะย้ายให้ไปอยู่ในไคลเอนต์โปรแกรมเลยได้ แต่ก็ไม่ควรจะทำ เพราะโปรแกรมของ RPCGEN นั้นจะเข้าใจได้ง่ายกว่าในระดับของ socket IPC

ฝั่ง Client

ในการเชื่อมต่อระหว่างไคลเอนต์และเซิร์ฟเวอร์แต่ละครั้งนั้น ไคลเอนต์จะต้องมีการสร้าง structure เฉพาะของตนไว้ (CLIENT) สำหรับควบคุมการติดต่อ โดยใน ONC RPC library นั้นจะมีดังนี้

clnt_create ( )

```
CLIENT * clnt_create (host, prognum, versnum, protocol)
        char *host;
        u_long prognum, versnum;
        char *protocol;
```

ใช้สำหรับสร้าง CLIENT ที่เชื่อมต่อกับเซิร์ฟเวอร์ขึ้นมาโดยระบุ host, program number, version และ protocol ที่ใช้ ใน ONC RPC 4.0 UDP จะส่งข้อมูลสูงสุดได้เพียง 8 K bytes เท่านั้น หากต้องการส่งมากกว่านี้ควรใช้ TCP หลังจากเรียกใช้ หาก portmapper ของโฮส ที่เรียกไปมีหมายเลขโปรแกรมนั้นอยู่จะส่งค่า CLIENT กลับมา หากไม่มีก็จะส่ง NULL กลับมา ส่วนเวอร์ชั่นนั้นหากไม่ตรงก็ยังไม่ error แต่จะไปมีผลใน cln_call ( )

clnt_destroy ( )

```
clnt_destroy (clnt)
        CLIENT *clnt;
```

ใช้สำหรับการยกเลิกการใช้ memory ของ CLIENT และหาก RPC มีการเปิดซ็อคเก็ตไว้ก็จะทำการปิดให้ด้วย

clnt_control ( )

clnt_control (clnt, request, info)

CLIENT *clnt;

int request;

char *info;

หลังจากสร้างการเชื่อมต่อด้วย clnt_create ( ) แล้ว จะใช้ clnt_control ( ) ในการเปลี่ยนแปลงหรือเรียกคุณสมบัติต่างๆของ CLIENT ที่ใช้ เช่น timeout socket, descriptor, สถานะต่างๆ ใช้ได้ทั้ง UDP และ TCP

clnt_call ( )

enum clnt_stat

clnt_call (clnt, procnum, inproc, in, outproc, out, timeout)

CLIENT *clnt;

u_long procnum;

xdrproc_t inproc, outproc;

char *in, *out;

struct timeval timeout;

ใช้สำหรับการติดต่อโดย CLIENT จะติดต่อไปยัง procnum โดยทำการ encode request in โดย inproc( ) จากนั้นจะรอ reply กลับมา แล้ว outproc( ) จะทำการ decode ออกมาเก็บไว้ใน out นอกจากนี้ยังสามารถตั้ง timeout ได้

ฝั่ง Server

ใน application_server.c นั้นไม่จำเป็นต้องเขียนโพรซีเดอร์ที่เกี่ยวกับ RPC เลย เพราะใน stub จะทำการควบคุมให้หมดแล้ว จึงเขียน code เฉพาะ procedure ที่เขียนไว้ใน protocol specification ก็พอ และจะใช้ SVCXPRT ในการ handle คล้ายกับ CLIENT ซึ่งจะมีดังนี้

svc_create ( )

SVCXPRT *svc_create (procnum, prognum, versnum, protocol)

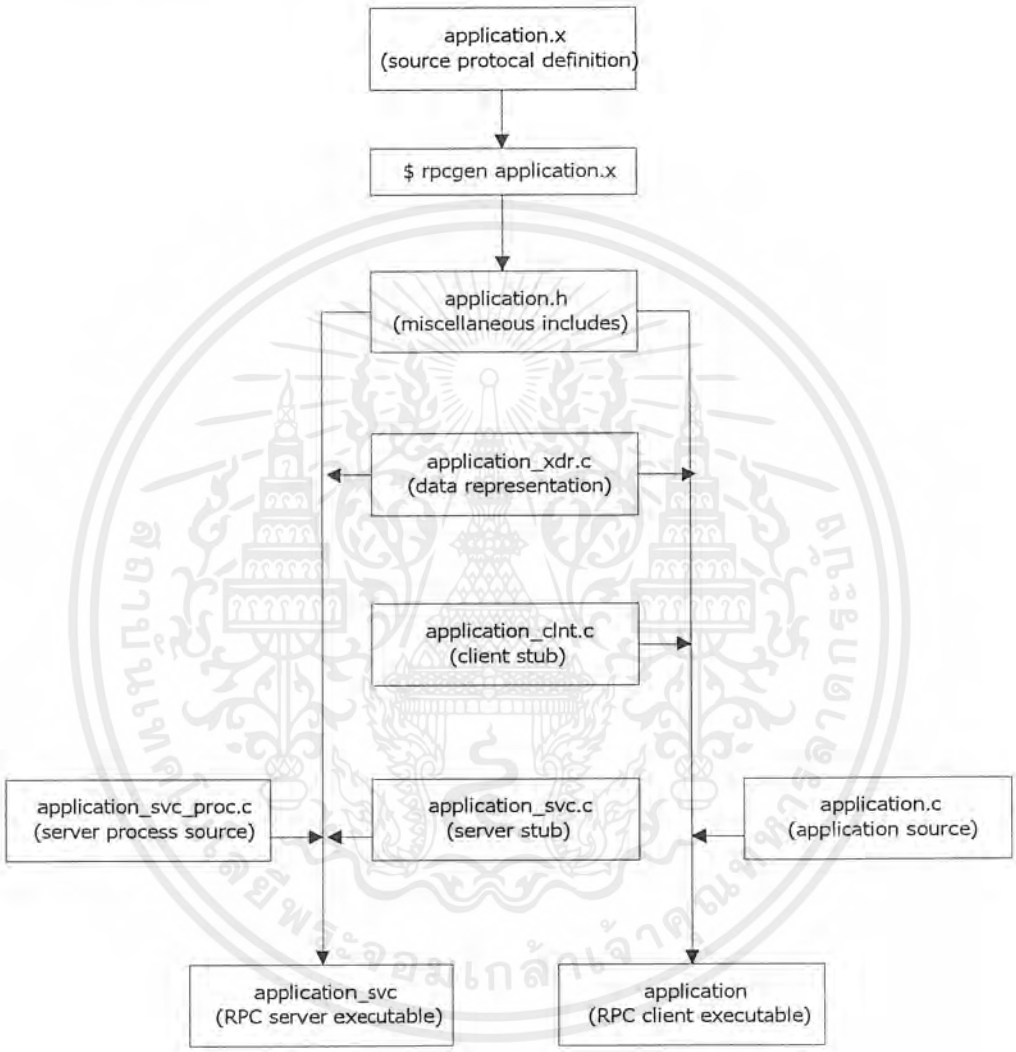
char *proc;

u_long prognum, versnum;

char *protocol;

ใช้สำหรับการ register procedure, program number, version, protocol แล้วหากไม่สามารถ register ได้ก็จะส่งค่า NULL กลับมา

2.6.5 การ Compile และการ Run โปรแกรม



รูปที่ 2.6.5-1 ขั้นตอนการคอมไพล์โดยใช้ rpcgen

เมื่อได้ compile RPC program แล้วจำเป็นต้อง link กับ stub and XDR filter ที่สร้างโดย RPCGEN ถ้าหากทำงานโดย SUN พวก XDR and RPC Library function จะเก็บอยู่ใน libc.a ซึ่งยังไม่ได้ link ให้เราสามารถทำการ link ได้โดยใช้คำสั่ง Makefile (เพื่อ link กับ lib ที่จำเป็นต่อ programนั้นๆ) หรืออาจทำเป็นไฟล์ ๆ ได้โดยใช้สั่ง >cc ซึ่งจะทำขั้นตอนต่างๆดังนี้

1. คอมไพล์ฝั่งไคลเอนต์

>cc –o application application.c application_xdr.c application_clnt.c

2. คอมไพล์ฝั่งเซิร์ฟเวอร์

>cc –o application_svc application_svc_proc.c application_svc.c application_xdr.c

เมื่อทำครบทุกขั้นตอนเราจะได้ไฟล์ที่สามารถทำการ executeได้โดยเราจะต้อง start up ที่ remote machine โดยใช้คำสั่ง >rsh –n remoteHost $cwd/application_svc & และจากนั้นเราสามารถทำการเรียกใช้โปรแกรมได้ที่ไคลเอนต์ ซึ่งจะได้ผลของการเรียกใช้โพรซีเดอร์จากฝั่งเซิร์ฟเวอร์และทำการส่งผลลัพธ์กลับมายังฝั่งไคลเอนต์

2.6.6 สถานะของ Server มีความสำคัญอย่างไรและมีสถานะอะไรบ้าง

เพราะว่าไคลเอนต์และเซิร์ฟเวอร์ทำงานต่างกันและแยกจากกันคนละที่ เพราะฉะนั้นจะเกิดอะไรขึ้นถ้าไม่สามารถติดต่อกันได้ หรือ server crash ระหว่างทำงานอยู่ แล้วไคลเอนต์จะทำอย่างไร ซึ่งการตอบสนองต่างๆจะขึ้นอยู่กับสถานะของเซิร์ฟเวอร์อย่างแรกคือ stateless คือจะไม่สนใจการติดต่อกันกับไคลเอนต์เลยว่าจะเป็นอย่างไร และ stateful คือเซิร์ฟเวอร์จะต้องทำงานได้ครบทุกอย่างตามที่ client request แต่ถ้าไม่มีการควบคุมสถานะนั้นไคลเอนต์จะต้องรอจนกว่าเซิร์ฟเวอร์จะหาย crash หรือรอจน time out ไปเอง และจะต้องมีการส่ง request ไปใหม่

2.6.7 ความผิดพลาดที่เกิดขึ้น

ปัญหาของ remote procedure calls แตกต่างจาก local calls เช่น failure ของไคลเอนต์และของเซิร์ฟเวอร์หรือกรณีที่ไคลเอนต์ไม่ได้รับ response ที่ต้องการ ซึ่งอาจเกิดจากกรณีต่างๆดังนี้

1. เน็ตเวิร์กช้ามากบวกกับเวลาที่ไคลเอนต์รอไม่ยาวนานพอ
2. ตัว initial message สูญหาย
3. เมื่อเซิร์ฟเวอร์ได้รับแมสเซสและทำการประมวลผลตาม request ที่รับมา แต่เกิด crashed ขึ้น
4. เมื่อเซิร์ฟเวอร์ทำการประมวลผลแล้วแต่เกิดการสูญหายขณะทำการส่งผลกลับไป (acknowledgment lost)

UDP เป็น connectionless protocol ไม่มีการตรวจสอบข้อผิดพลาดในการส่งและไม่สามารถทราบว่าแพ็คเก็จไปถึงปลายทางหรือยัง ตัวกลไกของ udp transport ของ ONC TI-RPC จะเตรียมการทำงานแบบ maybe semantics เพราะมันไม่สามารถรับรองว่า packet arrives (maybe semantics จะcall เพียงครั้งเดียวโดยเมื่อหมด period of time และจะไม่ทราบได้ว่าเกิดข้อผิดพลาดที่ขั้นตอนใด เช่น ขั้นตอน acknowledgment lost หรือ initial request lost)

TCP transport เตรียมการทำงานแบบ connection – oriented คือไม่มีข้อผิดพลาดของการ transmission ของข้อมูลการทำงานจะ tolerance จะเกิดการ retransmiting packet เมื่อได้รับ negative acknowledgment เท่านั้นโดยตัว TCP transport ของ ONC TI – RPC จะเตรียมการทำงานแบบ Most once

semantics (most once semantics จะทำงานกับ request ใดๆเพียงครั้งเดียวโดยไม่ทำงานซ้ำกับ request ที่เคยทำมาแล้วโดยมีการเก็บ request numbers ไว้ใน permanent storage)

2.7 ความรู้เรื่อง Automatic Transfer Machine

Automatic Transfer Machine หรือ ATM เป็นเครื่องที่ให้บริการสำหรับการถอนเงิน ฝากเงิน ถามยอดคงเหลือ และโอนเงินแก่ผู้ใช้บริการโดยผู้ใช้บริการจะต้องเป็นลูกค้าของธนาคารเจ้าของเครื่อง ATM หรือเป็นลูกค้าของธนาคารอื่นที่ธนาคารเจ้าของเครื่องตกลงยินยอมให้ใช้บริการได้ สำหรับการใช้บริการนั้นลูกค้าจะต้องทำบัตร ATM ที่มีบัญชีเงินฝากธนาคารนั้นอยู่ โดยจะได้หมายเลขบัตรและรหัสผ่านสำหรับใช้บริการ สำหรับรูปแบบและประเภทการให้บริการของ ATM แต่ละเครื่องจะไม่เหมือนกันขึ้นอยู่กับธนาคารเจ้าของเครื่องเป็นผู้กำหนดโดยส่วนมากจะประกอบไปด้วย

- บริการถามยอดคงเหลือ เป็นการให้บริการสำหรับถามยอดคงเหลือตามหมายเลขบัญชีเจ้าของบัตรที่ถือ
- บริการถอนเงิน เป็นการให้บริการการถอนเงินตามหมายเลขบัญชีเจ้าของบัตรที่ถือ
- บริการโอนเงิน เป็นการให้บริการการโอนเงินจากหมายเลขบัญชีเจ้าของบัตรที่ถือไปยังหมายเลขบัญชีที่ระบุ

ขั้นตอนการใช้บริการตู้ATM

การใช้บริการเครื่อง ATM มีขั้นตอนดังนี้

1. สอดบัตร ATM ยังตู้ ATM ที่ให้บริการ จากนั้นเครื่องจะทำการอ่านแถบแม่เหล็กหมายเลขบัตรและให้ใส่รหัสผ่านเพื่อส่งไปตรวจสอบสิทธิ์ยังเครื่องเซิร์ฟเวอร์
2. เมื่อรหัสผ่านถูกต้องแล้วสามารถที่จะเลือกใช้บริการประเภทต่างๆได้
3. ถ้าเลือกถามยอดคงเหลือจะแสดงยอดเงินคงเหลือของหมายเลขบัญชีนั้นออกมา
4. ถ้าเลือกบริการการถอนเงิน ตู้ ATM จะให้เราป้อนจำนวนเงินแล้วส่งไปยังศูนย์ประมวลผลเพื่อทำการตัดยอดแล้วทำการส่งผลลัพธ์กลับมา
5. ถ้าเลือกบริการการโอนเงิน ตู้ ATM จะให้เราป้อนหมายเลขบัญชีที่จะทำการโอนไปให้ และจำนวนเงินที่ต้องการจะโอน และจะแสดงผลลัพธ์ส่งกลับมายังผู้ใช้

2.8 การพัฒนาแอพพลิเคชั่นบนยูนิกซ์เท็กซ์โหมด

โปรแกรมจัดการจอภาพเป็นส่วนประกอบพื้นฐานของแอพพลิเคชั่นธุรกิจต่างๆโปรแกรมเหล่านี้จะจัดการอินพุทและเอาท์พุททางจอภาพ โปรแกรมจอภาพอาจเคลื่อนย้ายเคอร์เซอร์, พิมพ์เมนู, แบ่งพื้นที่จอภาพเป็นหน้าต่างหรือ แสดงจอภาพเพื่อช่วยการป้อนข้อมูลของผู้ใช้และเรียกข่าวสารจากดาต้าเบส

2.8.1 curses

curses(3x) คือไลบรารีของรูทีนที่สามารถใช้เพื่อเขียนโปรแกรมจัดการจอภาพบนระบบยูนิกซ์ รูทีนเหล่านี้เป็นฟังก์ชั่นภาษาซีและมาโคร หลายรูทีนก็ทำงานคล้ายคลึงในไลบรารีภาษามาตรฐาน ตัวอย่างเช่น มรรูทีน printw() ซึ่งทำงานคล้ายกับ printf(3s) มากและอีกรูทีนคือ getch() ซึ่งมีพฤติกรรมคล้ายกับ getch (3s) โปรแกรมสำหรับเครื่อง ATM ที่คุณเคยใช้อาจใช้ printw() เพื่อพิมพ์เมนูและใช้ getch() เพื่อรับคำสั่งฝากถอนเงินก็ได้ รวมทั้งโปรแกรมเอดิเตอร์ของระบบยูนิกซ์ เช่น เอดิเตอร์ vi(1) อาจใช้รูทีนเหล่านี้และรูทีนอื่นของ curses ก็ได้เช่นกัน

รูทีนของ curses ปกติจะอยู่ใน /usr/lib/libcurses.a เมื่อต้องการใช้รูทีนเหล่านี้จะต้องคอมไพล์โปรแกรมด้วยคำสั่ง cc(1) และอินคลูด –lcurses ในบรรทัดคำสั่งเพื่อลิงค์ไปยังอีดิเตอร์ที่ทำงานและโหลดรูทีนขึ้นมา

```
cc file.c –lcurses –o file
```

ชื่อ Curses นี้มาจากคำว่า cursor optimization ซึ่งรูทีนในไลบรารีนี้ทำงานดังชื่อนี้โดย cursor optimization จะลดจำนวนของเคอร์เซอร์ที่จะต้องวิ่งไปทั่วจอภาพเพื่ออัพเดทจอภาพ ตัวอย่างเช่น ถ้าออกแบบโปรแกรมเอ็ดดิเตอร์ด้วยรูทีน curses และต้องการแก้ไขข้อความ

```
Curses/terminfo is a great package for creating screens.
```

ให้เป็น

```
Curses/terminfo is the best package for creating screens.
```

โปรแกรมจะส่งข้อความออกมาเพียง the best ในตำแหน่งของ a great ส่วนตัวอักษรอื่นๆก้จะรักษาไว้คงเดิมจึงเห็นได้ว่า cursor optimization จะลดจำนวนข้อมูลที่ต้องส่งได้เป็นอย่างมาก

Cursor optimization ช่วยดูแลการอัพเดทจอภาพให้เหมาะสมกับเทอร์มินอล(terminal) ที่ curses รันอยู่ได้ หมายความว่าไลบรารี curses สามารถทำงานได้กับเทอร์มินอลหลายชนิดมันจะค้นหาดาต้าเบส terminfo เพื่อหาลักษณะที่ถูกต้องของเทอร์มินอล

เราจะมาดูกันว่า Cursor optimization สามารถช่วยผู้เขียนโปรแกรมและผู้ใช้งานได้อย่างไรประการแรกมันช่วยประหยัดเวลาที่ใช้อธิบายในโปรแกรมว่าคุณต้องการอัพเดทจอภาพอย่างไร ประการที่สอง คือมันช่วยประหยัดเวลาอัพเดทจอภาพขณะที่ผู้ใช้ใช้งาน ประการที่สามคือ มันช่วยลดภาระงานบนสายสื่อสารของระบบยูนิกซ์ที่ใช้เพื่ออัพเดท และประการที่สี่คือ ผู้เขียนโปรแกรมไม่ต้องกังวลเกี่ยวกับเทอร์มินอลหลากหลายชนิดที่โปรแกรมที่จะต้องรัน

รูป 2.8.1-1 เป็นโปรแกรม Curses ง่ายๆโปรแกรมนี้ใช้รูทีนพื้นฐานของ curses เพื่อย้ายเคอร์เซอร์ไปยังกลางจอเทอร์มินอล

```
#include<curses.h>
        main()
        {
                initscr();
                move(LINES/2-1,COLS/2-4);
                addstr("Bulls");
                refresh();
                addstr("Eye");
                refresh();
                endwin();
        }
```

รูปที่ 2.8.1-1 โปรแกรม curses

2.8.2 การใช้งาน curses

ชื่อ

curses() – แพ็คเกจเพิ่มประสิทธิภาพและจัดการจอภาพ CRT

SYSNOPSIS

#include <curses.h>

cc [flags] file... -lcurses [libraries]

คำอธิบาย

รูทีนเหล่านี้เสนอวิธีการอัพเดทจอภาพได้อย่างมีประสิทธิภาพ ก่อนที่จะใช้งานรูทีนใดๆ ที่เกี่ยวกับหน้าต่างหรือจอภาพ จำเป็นต้องอินนิเชียลรูทีน curses เสียก่อนด้วยคำสั่ง initscr() และก่อนจะออกจากโปรแกรมต้องใช้คำสั่ง endwin() และเมื่อต้องการรับตัวอักษรครั้งละหนึ่งตัวโดยไม่แสดงตัวอักษรให้เห็น (พบมากในโปรแกรมติดต่อกับผู้ใช้ที่เป็น interactive) หลังจากที่เรียก initscr() แล้วโปรแกรมควรใช้คำสั่ง

nonl(); cbreak(); noecho();

การใช้อินเตอร์เฟซของ curses อย่างสมบูรณ์ อนุญาตให้จัดการกับโครงสร้างข้อมูลแบบหน้าต่าง (windows) ซึ่งเป็นเหมือนอาร์เรย์สองมิติที่แทนการแสดงผลทั้งจอภาพหรือบางส่วนของจอภาพ ซึ่งมีค่าดีฟอลท์ของหน้าต่างทีเรียกว่า stdscr เตรียมให้อยู่แล้ว และสามารถสร้างหน้าต่างอื่นๆ อีกโดยใช้คำสั่ง newwin ก่อนที่หน้าต่างจะถูกใช้งานจะต้องประกาศชนิดตัวแปรเป็น "WINDOW *" ชนิดตัวแปร WINDOW นี้ถูกกำหนดไว้โดยโครงสร้างภาษาซี โครงสร้างของข้อมูลเหล่านี้จะถูกจัดการโดยฟังก์ชั่นดังที่จะกล่างต่อไป เช่นฟังก์ชั่นที่เป็นพื้นฐานที่สุด เป็นต้นว่า move() และ addch() (เวอร์ชั่นทั่วไปจะมีฟังก์ชั่นเหล่านี้ ถ้าชื่อขึ้นต้นด้วย w จะใช้เพื่อจัดการกับหน้าต่าง แต่สำหรับ stdscr ไม่จำเป็นต้องขึ้นต้นด้วย w) หลังจากนั้นต้องเรียก refresh() เพื่อบอกให้รูทีนสร้างจอภาพให้แสดงผลใหม่

Mimi-curses เป็นซับเซ็ต(subset) ของ curses ซึ่งไม่อนุญาตให้จัดการกับหน้าต่างมากกว่าหนึ่งหน้าต่างการเรียกเซ็ตย่อยนี้ ต้องเพิ่มออปชั่น –DMINICURSES ในคำสั่งของ cc(1) ซับเซ็ทนี้เล็กกว่าและทำงานได้เร็วกว่า curses แบบเต็ม

ถ้าตัวแปรสภาวะแวดล้อม TERMINFO ถูกกำหนดไว้โปรแกรมต่างๆที่ใช้ curses จะต้องตรวจข้อมูลที่กำหนดเป็นโลคอลเทอร์มินอลก่อนตรวจในพื้นที่มาตรฐาน ยกตัวอย่างเช่นถ้าพื้นที่มาตรฐานคือ /usr/lib/terminfo และ TERM ถูกเซ็ทให้เป็น vt100 ไฟล์ที่ถูกคอมไพล์จะพบได้ใน /usr/lib/terminfo/v/vt100 (ตัวอักษร v ดึงมาจากตัวอักษรตัวแรกของคำว่า vt100 เพื่อหลีกเลี่ยงการสร้างไดเร็กทรอรี่ขนาดใหญ่) แต่อย่างไรก็ตามถ้า TERMINFO ถูกเซ็ทไปยัง /usr/mark/myterms curses จะเริ่มตรวจใน /usr/lib/myterms/v/vt100 ก่อน ถ้าหากไม่พบก็จึงจะไปตรวจใน /usr/lib/terminfo/v/vt100 ลักษณะเช่นนี้เป็นประโยชน์แก่การพัฒนาการกำหนดตามต้องการ หรือเมื่อไม่กำหนดเพอร์มิชัน (permission) การเขียนให้เขียนใน /usr/lib/terminfo ให้

ฟังก์ชัน

ฟังก์ชันที่แสดงในตาราง 2.4-1 นี้เป็นเพียงส่วนหนึ่งของฟังก์ชันที่มีทั้งหมดของ curses ฟังก์ชันเหล่านี้เป็นฟังก์ชัน บางฟังก์ชันที่ใช้ในการพัฒนาโครงการระบบถอนเงินอัตโนมัติ ฟังก์ชันอื่นๆสามารถหาเพิ่มเติมได้ในคู่มืออ้างอิงการเขียนโปรแกรมบนยูนิกซ์ และในไฟล์ help ของระบบยูนิกซ์ รายชื่อฟังก์ชันที่แสดงนี้เรียงลำดับตามตัวอักษร ฟังก์ชันที่มีเครื่องหมายดอกจันเป็นฟังก์ชันที่สามารถเรียกได้โดย Mini-curses

ชื่อฟังก์ชัน	ความหมาย
Box(win,vert,hor)	วาด box รอบของของ win โดย vert และ hor เป็นตัวอักษรที่ใช้เป็นขอบตามแนวตั้งและแนวนอนของ box
Clear()	ลบข้อความทั้งหมดบน strscr
Delwin(win)	ทำลาย win
Echo()*	เซ็ตโหมด window
Endwin()*	จบโหมด window
initscr()*	อินนิเชียลจอภาพ
keypad(win,bf)	ทำให้ keypad ของอินพุททำงาน
newwin(lines,cols,begin y,begin x)	สร้าง window ใหม่
nl()*	เซ็ทการขึ้นบรรทัดใหม่
noecho()*	เซ็ทให้โหมด echo ไม่ทำงาน
nonl()*	เซ็ทให้การขึ้นบรรทัดใหม่ไม่ทำงาน
waddstr(win,str)	เพิ่มประโยคให้ win
wattroff(win,attrs)	ปิดการแสดงตามแบบตัวอักษร attrs ให้ win
wattron(win,attrs)	เปิดการแสดงตามแบบตัวอักษร attrs ใน win

wclear(win)	ลบข้อความทั้งหมดใน win
Werase(win)	ทำลาย win
Wgetch(win)	รับตัวอักษรผ่าน win
Winsertln(win)	แทรกบรรทัดใน win
Wmove(win,y,x)	เซ็ทโคออร์ดิเนทปัจจุบัน(y,x) บน win
Wprintw(win,fmt,arg1,arg2,...)	printf() บน win
Wcanw(win,fmt,arg1,arg2,...)	Scanf() ผ่าน win
Wefresh(win)	ทำจอภาพให้เป็นเหมือน win

ตาราง 2.8.2-1 บางฟังก์ชั่นของ curses

แอททริบิวท์

ค่าวีดีโอแอททริบิวท์ในตาราง 2.8.2-2 ต่อไปนี้สามารถผ่านไปยังฟังก์ชั่น attron(), attroff() และ attrset()

ชื่อแอททริบิวท์	ความหมาย
A_STAANDOUT	โหมดที่เห็นได้ชัดเจนที่สุดของเทอร์มินแล
A_UNDERLINE	ขีดเส้นใต้
A_REVERSE	แสดงกลับสีของวีดีโอ
A_BLINK	กระพริบ
A_DIM	สว่างปานกลาง
A_BOLD	สว่างมาก หรือตัวหนา
A_BLANK	ว่าง(ไม่สามารถมองเห็น)
A_PROTECT	ป้องกัน
A_ALTCHARSET	ชุดตัวอักษรที่เลือก

ตาราง 2.8.2-2 ตารางแสดงแอททริบิวท์ใน curses

2.8.3 โครงสร้างพื้นฐานของโปรแกรม curses

1. ก่อนที่จะเริ่มใช้ curses เราควรทำการโปรเซสบรรทัดคำสั่งเสียก่อน ยกตัวอย่างเช่นถ้าผู้ใช้ป้อน อาร์กิว เมนต์ในบรรทัดคำสั่งไม่ถูกต้อง โปรแกรมควรจะพิมพ์ข้อความไปยัง stderr โดยใช้ fprintf (3) และออกจากโปรแกรมไม่ต้องเข้าสู่โหมด curses จนกว่าโปรแกรมจะพร้อม
2. โครงสร้างภายในและตัวแปรหลายๆตัวจำเป็นต้องถูกตั้งค่าก่อนที่ curses จะถูกใช้ ดังนั้นจะต้องอินนิเชียล curses โดยปกติจะใช้คำสั่ง initscr() เพื่อเตรียมหน้าต่าง curscr และ stdscr, อินนิเชียลเทอร์มินอล, และโปรแกรมจึงเข้าสู่โหมด in-curses

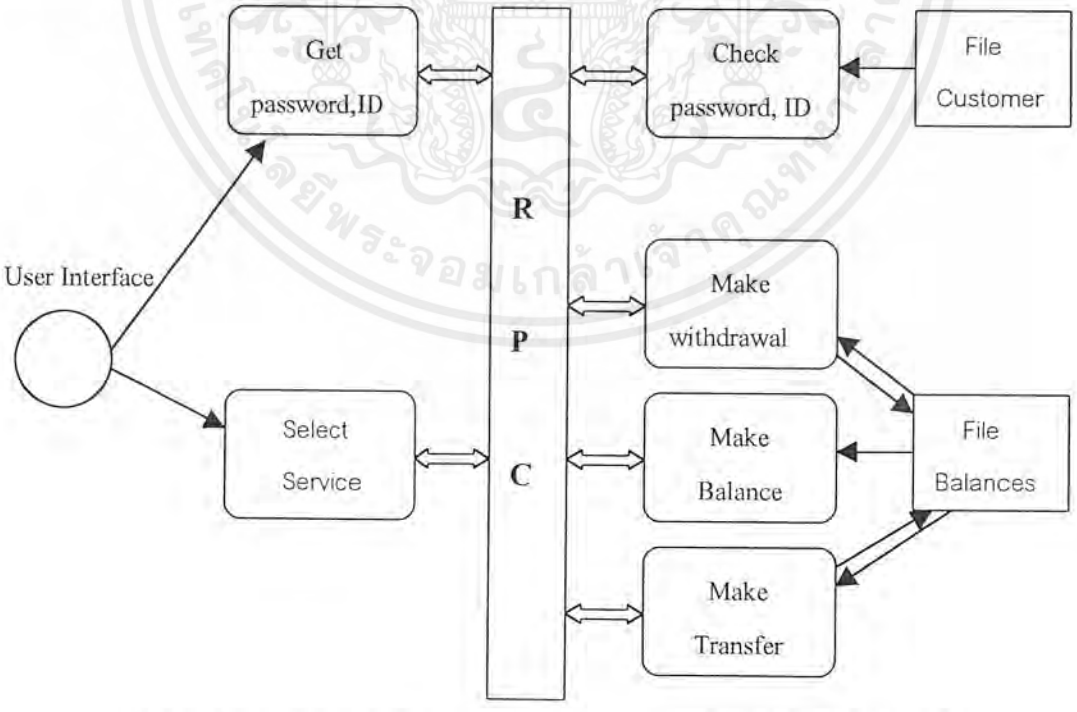
3. ถ้าต้องการตั้งค่าโหมด I/O พิเศษในเทอร์มินอลไดร์เวอร์จะต้องทำหลังจากอินนิเชียลแล้ว โดยปกติแล้วโหมดเหล่านี้จะถูกเซ็ทเพียงครั้งเดียว และยังคงไม่แก้ไขอีกตลอดเวลาที่โปรแกรมทำงาน ยกตัวอย่างเช่น การเซ็ทเทอร์มินอลให้เข้าสู่โหมด raw , ปิดการ echoing และอื่นๆ
4. ขณะนี้โปรแกรมทำงานในโหมด curses และทำงานต่อไปได้โดยเครื่องมือของ curses ยกตัวอย่าง เช่น สร้าง/ทำลาย หน้าต่าง, เพิ่มตัวอักษรในหน้าต่าง, รับอินพุทจากคีย์บอร์ดและอื่นๆ
5. เมื่อโปรเซสสมบูรณ์และโปรแกรมพร้อมที่จะรีเทิร์นกลับไปยังเชลล์(หรือ โปรเซสที่เรียก) เทอร์มินอลจะต้องกลับไปยังโหมดการทำงานดั้งเดิมคือโหมดปัจจุบันก่อนที่จะเข้ามาในโปรแกรม ขั้นตอนนี้ทำโดย endwin()
6. ในที่สุด ก็ออกจากโปรแกรม

บทที่ 3
การออกแบบและโครงสร้างของโปรแกรม

3.1 โครงสร้างการทำงานโดยรวมของระบบจำลอง ATM

โครงสร้างของการจำลองการถอนหรือโอนเงินของตู้ ATM จะประกอบไปด้วยส่วนหลักๆสองส่วน คือส่วนของโปรเซสที่อยู่ด้านเซิร์ฟเวอร์และอีกโปรเซสที่อยู่ทางด้านฝั่งไคลเอนต์ เสมือนกับการติดต่อกันระหว่างตู้ ATM ที่ให้บริการกับส่วนที่เก็บข้อมูลและส่วนประมวลผลกลางที่ธนาคาร โดยทั้งสองโปรเซสจะอยู่คนละเครื่องกัน ที่ทางด้านฝั่งเซอร์เวอร์จะทำหน้าที่เกี่ยวกับการประมวลผล และอัพเดทข้อมูลในไฟล์ดาต้าเบส โดยจะสามารถให้บริการแก่ไคลเอนต์หลายๆตัวเข้ามาใช้บริการพร้อมกันได้ ส่วนทางด้านฝั่งไคลเอนต์จะทำหน้าที่รับอินพุทมาจากผู้ใช้งานและทำการแสดงเอาท์พุทให้ผู้ใช้ดู โดยใช้วิธีการติดต่อสื่อสารแบบ RPC ลักษณะการทำงานเช่นนี้จะเป็นลักษณะการทำงานที่เรียกว่า Client/Server โดยเป็นการกระจายการทำงานของตัวแอพพลิเคชัน โดยการคำนวนทุกอย่างจะอยู่ที่ฝั่งเซอร์เวอร์ทั้งสิ้น ส่วนทางฝั่งไคลเอนต์จะมีหน้าที่เพียงเป็นอินเทอร์เฟสที่ใช้รับคำสั่งและแสดงผลเท่านั้นเอง

โดยทางด้านตู้ ATM จะเป็นเสมือนตัวไคลเอนต์ ทำหน้าที่ในการ แสดงการอินเตอร์เฟส สำหรับติดต่อกับผู้ใช้ จะมีการรับคำสั่งสำหรับส่งไปประมวลผลยังฝั่งเซิร์ฟเวอร์ และรับเอาท์พุทจากเซิร์ฟเวอร์มาแสดงผล ส่วนประมวลกลางสำนักงานใหญ่หน้าที่เป็น เซิร์ฟเวอร์รับคำสั่งการขอดูยอดคงเหลือ การถอนเงิน และการโอนเงินจากตู้ ATM ต่างๆ และทำการประมวลผลกับข้อมูลส่วนกลาง แล้วส่งผลลัพธ์ที่ได้ไปยังไคลเอนต์ แสดงดังรูปที่ 3.1-1

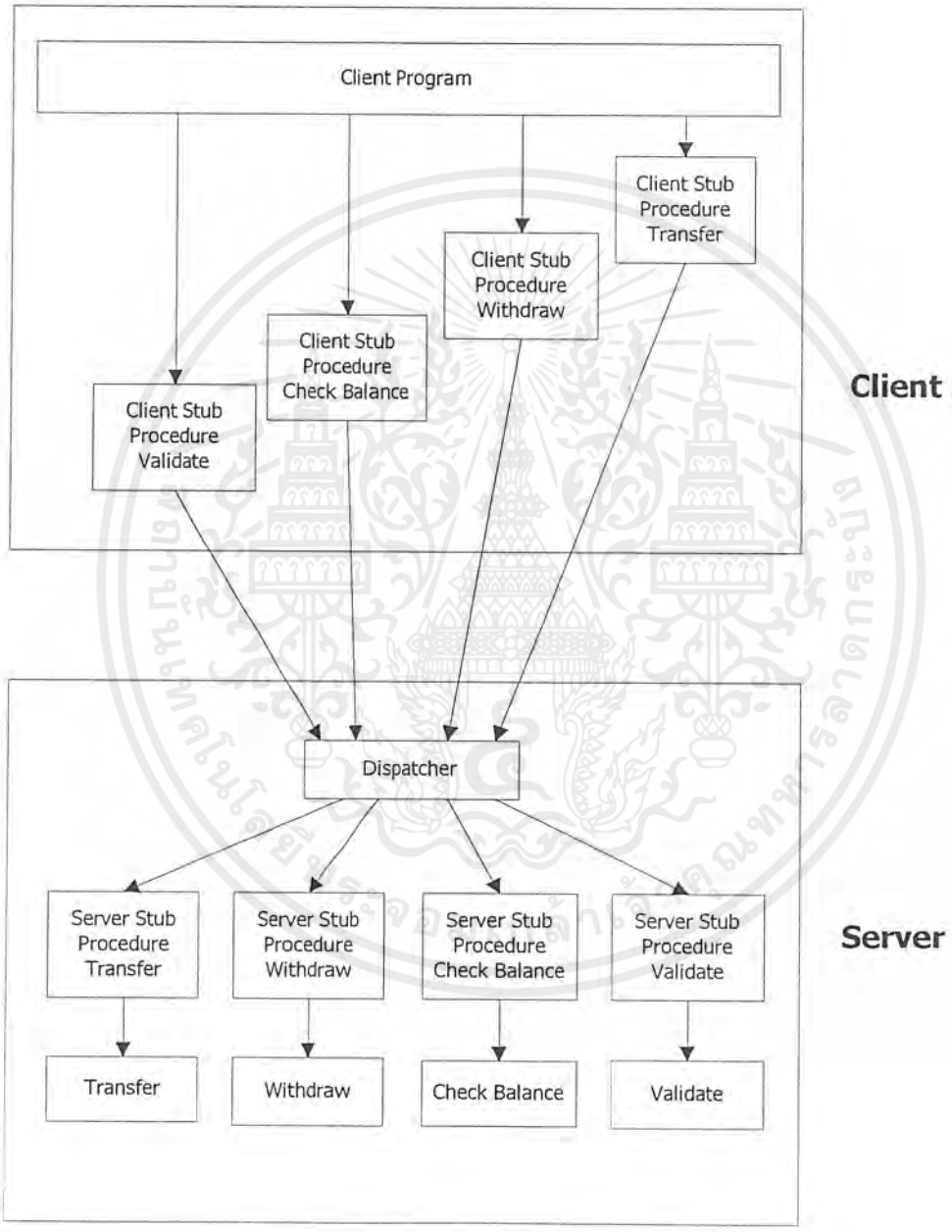


รูปที่ 3.1-1 แสดงโครงสร้างการทำงานโดยรวมของระบบจำลอง ATM ผ่านกลไก RPC

3.2 โครงสร้างการทำงานของโปรแกรม

3.2.1 การออกแบบการติดต่อกันระหว่าง client และ server โดย RPC

ในการออกแบบโปรแกรมจะต้องประกอบไปด้วยโปรแกรมฝั่งไคลเอ็นต์และโปรแกรมฝั่งเซอร์ฟเวอร์ โดยโปรแกรมฝั่งไคลเอ็นต์นั้นต้องระบุหมายเลขเครื่องฝั่งเซิร์ฟเวอร์ , หมายเลขของโปรแกรมที่เรียกใช้ และโพรซีเดอร์ที่จะเรียกใช้ และโปรแกรมในฝั่งเซิร์ฟเวอร์จะต้องประกอบด้วย dispatcher routine, โพรซีเดอร์ ฝั่งเซิร์ฟเวอร์ และ server-stub ดังแสดงในรูปที่ 3.2.1-1



รูปที่ 3.2.1-1 การเรียกใช้ remote procedure ด้วยวิธี rpc

โดย dispatcher จะต้องเข้าใจว่าหมายเลขของ remote procedure จะตอบสนองกับ server-stub อย่างไรและใช้การตอบสนองนั้นสำหรับการส่งให้ stub ที่สัมพันธ์เมื่อมีการเรียก remote procedure call แต่ละครั้ง จากรูปที่ 3.2.1-1 เมื่อไคลเอ็นต์ request ไปที่เซิร์ฟเวอร์ dispatcher จะทำการตัดสินใจในการเรียกใช้โพรซีเดอร์ตามที่ได้รับแมสเซสจากไคลเอ็นต์

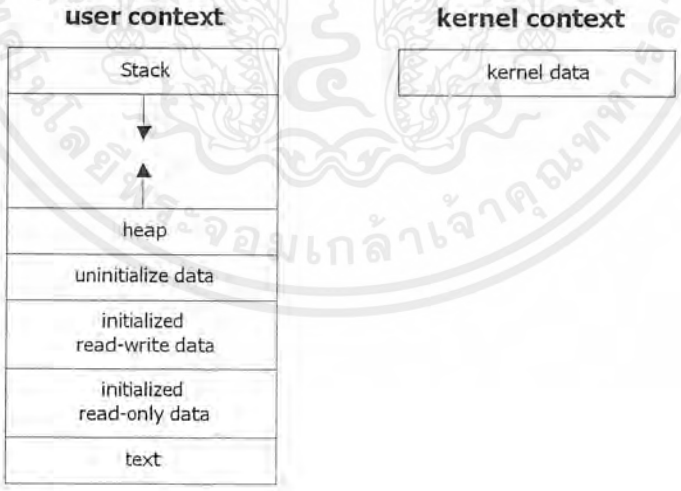
3.2.2 โปรเซสและการอินเตอร์รัพโปรเซส

โปรเซสและสัญญาณ (การอินเตอร์รัพโปรเซส) เป็นออบเจ็คพื้นฐานสำหรับระบบปฏิบัติการยูนิกซ์ โดยจะควบคุมงานทุกอย่างบนระบบ การจัดการโปรเซสนั้นจะประกอบไปด้วยการสร้างและหยุดการทำงานของโปรเซสจากโปรแกรม การส่งข้อความ (ระบุสถานะ) ของโปรเซสไปยังโปรเซสอื่น

โปรเซสนั้นจะเป็นชุดคำสั่งในหน่วยความจำที่กำลังรันอยู่ในขณะนั้นโดยมีชุดควบคุมการทำงาน 1 ชุดเป็นของตัวเอง โดยการรันนั้นจะต้องใช้งานรีซอร์สของระบบไปส่วนหนึ่ง ระบบที่เป็น Multitasking เช่นยูนิกซ์ สามารถจะรันโปรแกรมเพียงโปรแกรมเดียว ได้หลายโปรเซสในเวลาเดียวกัน เช่น เมื่อมีผู้ใช้งาน 10 คนกำลังใช้งานเท็กซ์อิดิตเตอร์อยู่ เป็นต้น นอกจากนี้ในฐานะที่เป็นระบบ multi-user หนทางเดียวที่จะสร้างโปรเซสใหม่ในระบบยูนิกซ์ คือการใช้ฟังก์ชั่นระดับต่ำ fork

ดังนั้นโปรเซสจะประกอบด้วยโค้ดของโปรแกรม ข้อมูลของโปรเซส ตัวแปร หมายเลขไฟล์ (file descriptor) และสภาพแวดล้อมการทำงานของโปรเซส ข้อมูลบางอย่าง โปรเซสสามารถใช้งานร่วมกันได้ เช่นโค้ดของโปรแกรมที่เหมือนกัน หรือไลบารีที่ใช้งานร่วมกัน

โครงสร้างของโปรเซส



รูปที่ 3.2.2-1 โครงสร้างของโปรเซส

ในส่วน user context จะประกอบไปด้วยส่วนต่างๆดังนี้

- text จะเก็บคำสั่งระดับต่ำ (machine instruction) ของเครื่องที่จะรันโดยฮาร์ดแวร์ ปรกติจะกำหนดให้มี เพอร์มิสชั่นเป็น read only ดังนั้นโปรเซสไม่สามารถจะแก้ไขข้อมูลในส่วนนี้ได้ นอกจากนี้บางคำสั่งยังสามารถใช้งานร่วมกันได้อีกด้วย
- data เก็บข้อมูลที่จะใช้ในโปรแกรม จะแบ่งเป็นสามส่วนด้วยกัน คือ uninitialized data, initialized read-write data และ initialized read only data
- heap จะใช้งานในขณะรันโปรเซส เพื่อกำหนดพื้นที่หน่วยความจำให้เพียงพอแก่ความต้องการใช้งาน
- stack สำหรับควบคุมการใช้งานตัวแปร การเรียกใช้ฟังก์ชั่นและการส่งค่ากลับ counter บันทึกจุดที่กำลังรัน(execution thread)

พื้นที่ว่างระหว่าง heap และ stack จะทำให้ระบบปฏิบัติการ สามารถใช้งานออปเจ็กต์ทั้งสองได้อย่างมีประสิทธิภาพ (dynamically)

kernel context จะควบคุมการใช้งานโดยเคอร์เนลนั้น เพื่อเก็บข้อมูลที่จำเป็นสำหรับติดตามการทำงานของโปรเซส หรือเพื่อเริ่มรัน หยุดรันโปรเซส เช่นตำแหน่งของโปรเซส ขนาดของโปรเซส โปรเซสโดยทั่วไปไม่สามารถจะแอ็คเซสข้อมูลในส่วนนี้ได้ ข้อมูลจะเก็บไว้ในโครงสร้างข้อมูลของโปรเซส (process table) เช่น

process identifier หรือ PID ปกติจะมีค่าตั้งแต่ 2-32000 (หมายเลขโปรเซสของระบบ จะมีค่าตั้งแต่ 1-100, reserved pid) ดังนั้นหมายเลขโปรเซสที่ใช้งานจึงมีค่าตั้งแต่ 101

parent process identifier(PPID) , Real User ID(user ID), Real Group ID(group ID), ลำดับความสำคัญของโปรเซส, เทอร์มินอล สถานะของโปรเซสในขณะนั้น

โดยปกติในระบบ multi-tasking แต่ละโปรเซสจะมีช่วงเวลาการรันเป็นของตัวเอง(time slices) ระบบยูนิกซ์จะมีตารางกำหนดการทำงานของโปรเซสในแต่ละ time slice การกำหนดนี้จะอยู่บนพื้นฐานของลำดับความสำคัญของโปรเซส โปรเซสที่มีลำดับความสำคัญสูง(เลขระบุความสำคัญเป็นลบ) จะรันบ่อยกว่าโปรเซสที่มีลำดับความสำคัญน้อย(เลขระบุลำดับความสำคัญเป็นบวก)

การสร้างโปรเซสขึ้นมาใหม่

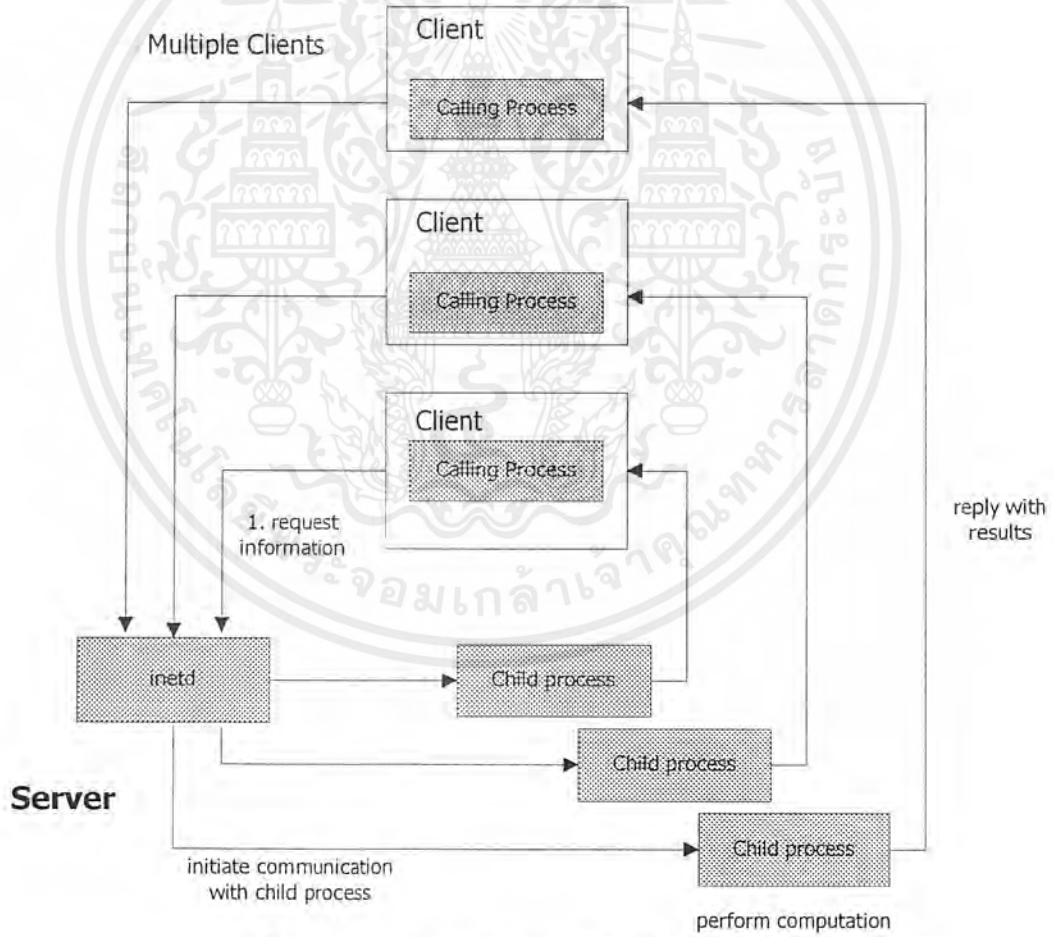
ถ้าต้องการให้โปรแกรมทำงานมากกว่าหนึ่งโปรเซส ในเวลาเดียวกัน ก็ต้องสร้างโปรเซสอื่นขึ้นมาใหม่เช่น init ที่ทำงานเมื่อตอนบูต ในขณะที่ฟังก์ชั่นตระกูล exec จะเป็นเพียงการเปลี่ยนชุดคำสั่งที่จะรันในโปรเซส

โดยการใช้ฟังก์ชั่นระดับต่ำ fork เพื่อสร้างโปรเซสลูกใหม่ โปรเซสที่สร้างขึ้นใหม่จะมีคุณสมบัติเหมือนกับโปรเซสต้นฉบับทุกประการ แต่มีค่า PID เป็นของตนเอง และมีเนื้อที่ในการทำงานเป็นของตนเองเช่น หน่วยความจำ เวลาในการใช้งานซีพียู สภาวะในการทำงาน

โดยทั่วไปฟังก์ชั่น fork จะใช้งานกรณีที่ โปรเซสต้องการที่จะเอ็กซ์คิวท์ โปรแกรมใหม่ เนื่องจากวิธีการสร้างโปรเซส จะมีอยู่วิธีเดียวเท่านั้น คือการใช้ฟังก์ชั่น fork ดังนั้นโปรเซสจะต้อง fork เพื่อสร้างโปรเซสใหม่ จากนั้นค่อยใช้ฟังก์ชั่น exec เพื่อรันโปรแกรม

3.2.3 การทำ concurrency control ด้วยวิธี inetd

เนื่องจากเป็นการจำลองการทำงานของเครื่อง ATM ดังนั้นต้องสามารถที่จะรองรับการทำงานแบบ หลายไคลเอนต์ได้แต่เนื่องจากมีหลายวิธีที่จะสามารถทำได้แต่ที่เลือกใช้วิธีของ inetd ซึ่งแสดงดังรูปที่ 3.2.3-1 เนื่องจากเป็น โปรเซส deamon ที่มากับยูนิกซ์ ทำให้ง่ายต่อการเรียกใช้เพียงแค่กำหนดโปรแกรมฝั่งเซิร์ฟเวอร์ไว้ในไฟล์ /etc/inetd.conf โดยโปรเซส inetd จะรอคอยให้บริการสำหรับการร้องขอแบบหลายไคล์เอนต์ และจะทำการเรียกฟังก์ชัน fork และ exec เพื่อให้โปรแกรมบนเครื่องเซิร์ฟเวอร์จริงๆทำงานนั่นก็คือ โปรแกรมให้บริการโอนเงิน บริการถอนเงิน บริการเช็คยอด และตรวจสอบรหัสผ่านนั่นเองสำหรับการร้องขอแต่ละครั้ง



รูปที่ 3.2.3-1 การทำ mutiple client ด้วยวิธีของ inetd

ดังนั้นจะต้องทำการ config inetd ก่อนโดยจะต้องกำหนดในไฟล์ /etc/inetd.conf ในการจะ config ไฟล์ /etc/inetd.conf นั้นจะต้องทำความเข้าใจก่อนว่าในไฟล์นั้นแสดงรายการอะไรบ้างและมีไฟล์อะไรที่

สัมพันธ์ที่จะต้องกำหนดเพิ่มเติม โดยไฟล์ /etc/inetd.conf เป็นไฟล์ที่ถูกอ่านโดยโปรแกรม inetd ซึ่งเป็น deamon process ซึ่งภายในไฟล์จะแสดงรายการของซ็อคเก็ตที่ปรากฏอยู่ในไฟล์ /etc/services และบางรายการจะปรากฏอยู่ในไฟล์ /etc/rpc ซึ่งแต่ละรายการในไฟล์ /etc/inetd.conf มีรูปแบบดังนี้

{ service-name socket-type protocol wait-status user

server-pathname server-arguments }

เพราะฉะนั้น จะต้องทำการ config ไฟล์ /etc/inetd.conf โดยเพิ่มเข้าไปในไฟล์ดังนี้

serv stream tcp nowait root /tmp/project/serv serv

อธิบายได้ดังนี้

- serv เป็นชื่อที่จะเรียกใช้ซึ่งจะต้องปรากฏอยู่ในไฟล์ /etc/services ด้วย
- ส่งข้อมูลแบบสตรีม
- โปรโตคอลเป็นแบบ tcp ซึ่งจะปรากฏในไฟล์ /etc/protocol
- โปรแกรมเป็นแบบ nowait สำหรับโปรโตคอลแบบ tcp
- login ด้วย root
- /tmp/project/serv เป็น path ที่อยู่ของโปรแกรมเซิร์ฟเวอร์

จากนั้นจะต้องทำการ config ไฟล์ /etc/rpc โดยเพิ่มหมายเลขโปรแกรมเข้าไปในไฟล์ดังนี้

serv 123456789

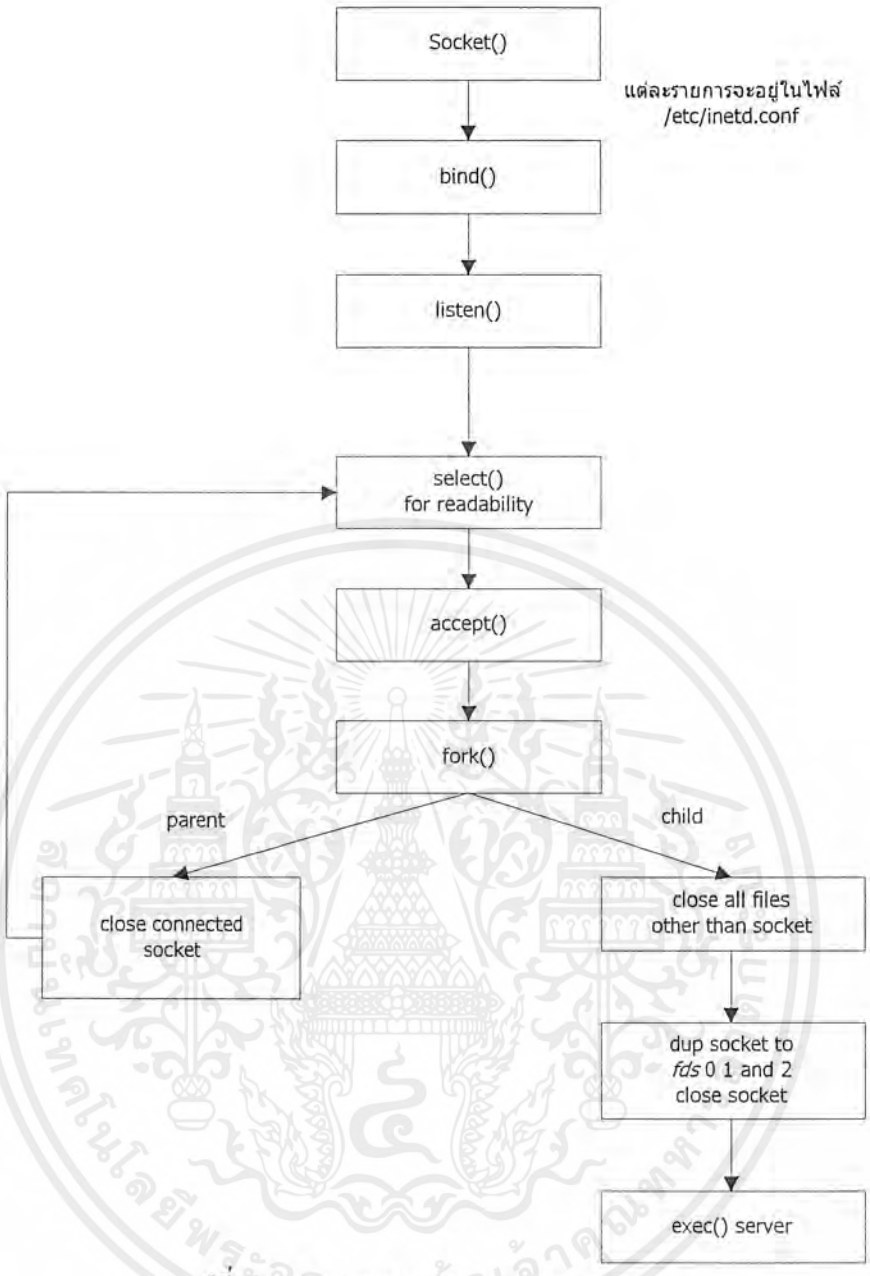
- 123456789 เป็นหมายเลขโปรแกรมของเซิร์ฟเวอร์โปรแกรม

และทำการ config ไฟล์ /etc/services โดยเพิ่ม port และโปรโตคอลที่ใช้เข้าไปในไฟล์ดังนี้

- **serv 810/tcp**

หลังจาก config ไฟล์ทั้งหมดแล้วสามารถที่จะเรียกใช้ได้เลย โดยโปรแกรมฝั่งไคล์เอ็นต์จะเรียกใช้โปรแกรมฝั่งเซอร์เวอร์นั้น ก็จะต้องเรียกผ่านโปรแกรม inetd ซึ่งวิธีการเรียกนั้นก็เรียกเหมือนเดิมก่อนที่จะเพิ่มการทำ concurrent สำหรับขั้นตอนการทำงานของโปรแกรม inetd แสดงดังรูปที่ 3.2.3-2 ดังนี้

1. เมื่อเริ่มต้นทำงานจะอ่านในไฟล์ /etc/inetd.conf และจะสร้างซ็อคเก็ตที่เป็นแบบ stream เพื่อให้บริการตามที่กำหนดในไฟล์
2. ในแต่ละซ็อคเก็ตที่ถูกสร้างขึ้นก็จะ bind เพื่อหาแอดเดรสให้แก่ซ็อคเก็ตนั้น ซึ่งหมายเลขพอร์ตที่ได้ก็เป็นหมายเลขที่เก็บอยู่ในไฟล์ /etc/services นั่นเอง โดยชื่อโปรแกรมและโปรโตคอลจะถูกส่งไปยังฟังก์ชัน getservbyname เพื่อจอง port ให้แก่ bind



รูปที่ 3.2.3-2 แสดงการทำงานของ inetd deamon

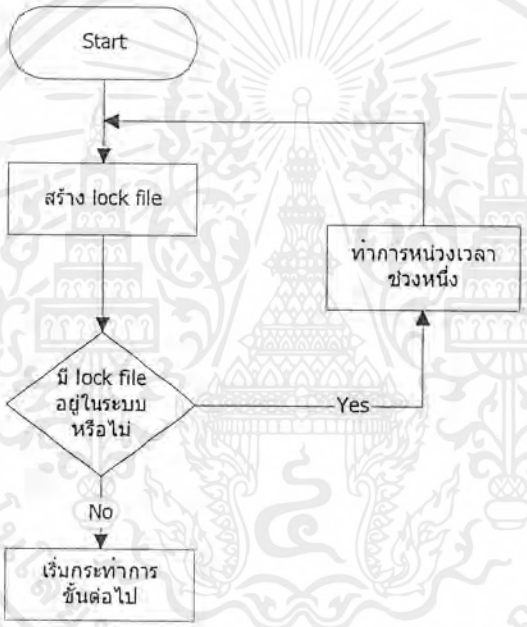
3. ฟังก์ชัน listen ทำงานเพื่อกำหนดการรับการเชื่อมต่อบนซ็อคเก็ตและจัดการกับความยาวของคิวที่จะรับเข้ามา
4. ฟังก์ชั่น select ทำงานเพื่อรอให้ซ็อคเก็ตแรกพร้อมสำหรับการอ่านเมื่อการ request เดินทางมาถึง โดยในจุดนี้ inetd deamon ก็จะรอคอยให้ ฟังก์ชัน select ทำงานเสร็จ
5. เมื่อซ็อคเก็ตพร้อมสำหรับการอ่านก็จะเรียกฟังก์ชัน accept เพื่อรับการเชื่อมต่อ
6. inetd deamon จะ fork โปรเซสลูกเพื่อให้บริการในการ request แต่ละครั้ง โดยที่โปรเซสลูกจะ dup socket เป็น fds 0,1 และ 2 และปิดซ็อคเก็ตเริ่มต้น จากนั้นโปรเซสลูกก็จะเรียกฟังก์ชั่น exec เพื่อประมวลผลโปรแกรมที่ได้ระบุไว้ตาม path

7. โปรเซสแม่จะต้องปิดซ็อคเก็ตโดยจะกลับไปเรียกฟังก์ชัน select อีกครั้งและรอคอยให้ซ็อคเก็ตต่อไปพร้อมสำหรับการอ่าน

3.2.4 การทำ lock file สำหรับจัดสรรการใช้ทรัพยากร

เนื่องจากยูนิกซ์เป็นระบบ multi-user , multi-tasking ดังนั้นการจะป้องกันไม่ให้โปรเซสอื่นแอ็คเซสไฟล์เป็นสิ่งจำเป็นหรือป้องกันไม่ให้โปรเซสอื่นทำงานบนไฟล์ขณะที่กำลังมีการเปลี่ยนแปลง (transient state) เช่น การป้องกันการอ่านหรือเขียนข้อมูลในไฟล์ ในขณะที่โปรเซสอื่นกำลังอ่านหรือเขียนข้อมูลอยู่ เป็นต้น

เพื่อจัดการระบบให้เป็นไปตามวัตถุประสงค์ดังกล่าว ยูนิกซ์มีวิธีการหลายวิธี วิธีที่ง่ายคือสร้าง locked file เพื่อที่จะเข้าควบคุมไฟล์ทั้งหมดแสดงดังรูปที่ 3.2.4-1



รูปที่ 3.2.4-1 การทำ lock file ในระบบ

ตัวอย่างของการสร้าง lock file เพื่อป้องกันไม่ให้มีการเข้าถึงไฟล์ขณะที่มีโปรเซสหนึ่งกำลังทำงานอยู่

```
fildes = open(lock_file, O_RDWR | O_CREAT | O_EXCL,0444);
        if (fildes == -1) {
                transfer_result.Flag=-1;
                printf("%d",transfer_result.Flag);
                printf(" %d - Lock already present\n",getpid());
                return(&transfer_result);
```

```
        }
    else {
            printf(" %d - I have exclusive access\n",getpid());
        }
```

การสร้างไฟล์ให้เป็น locked file จะใช้ฟังก์ชันระบบ open และใช้แฟลก O_CREAT ร่วมกับ O_EXCL ซึ่งจะเช็คว่ามีไฟล์ชื่อนั้นอยู่หรือไม่ จากนั้นจะสร้างไฟล์ในกระบวนการที่เรียกว่า atomic operation

3.2.5 การกำหนดเวลา time-out

หลังจากที่ไคลเอนต์ส่ง request ไปที่เซิร์ฟเวอร์แล้วไคลเอนต์จะรอรับแมสเซสที่จะ reply จากเซิร์ฟเวอร์ กลับมาซึ่งปกติจะ default เป็น 25 วินาที เราสามารถกำหนดเวลา time out ในกรณีที่ไคลเอนต์ไม่ได้รับแมสเซสจากเซิร์ฟเวอร์เพื่อให้ทำการส่งใหม่อีกครั้งจนกว่าไคลเอนต์จะได้รับแมสเซสจากเซิร์ฟเวอร์หรือครบจำนวนครั้งที่กำหนด ในการกำหนดเวลา time out นั้น สามารถเรียกใช้ฟังก์ชั่น clnt_control() ซึ่งมีตัวอย่างการใช้ดังรูปที่ 3.2.5-1

```
struct timeval tv;
CLIENT *cl;
cl = clnt_create("161.246.5.240", ATMPROG, ATNVERS, "tcp");
if (cl == NULL) {
    exit(1);
}
tv.tv_sec  = 60; /* change timeout to 1 minute */
tv.tv_usec = 0;
clnt_control(cl, CLSET_TIMEOUT, &tv);
```

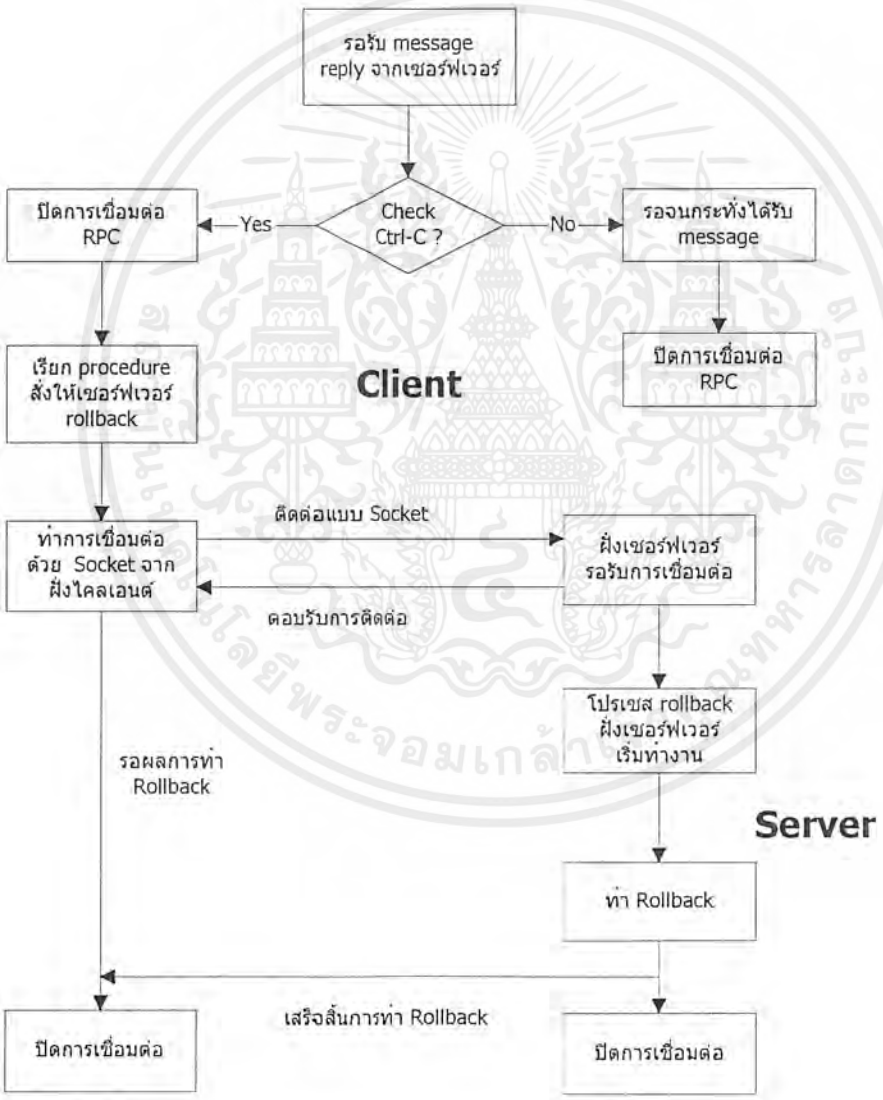
รูปที่ 3.2.5-1 ส่วนของโปรแกรมที่กำหนดเวลา time-out

ถ้าใช้การติดต่อแบบ UDP โดยปกติแล้วไคลเอนต์จะทำการ retry request ให้ทุกๆ 4 วินาที ฟังก์ชัน clnt_control นี้จะไปเปลี่ยนค่า TIMEOUT ในฟังก์ชั่น clnt_call ซึ่งเป็นฟังก์ชันระดับล่างของ RPC

3.2.6 การทำ Rollback

สำหรับกรณีที่ทำ transaction ใดๆแล้วไคลเอนต์รอรับ reply แมสเซสจากเซิร์ฟเวอร์ซึ่งใช้เวลา

นานมากแต่น้อยกว่าเวลา time-out ที่กำหนดไว้ โดยปัญหาอาจจะเกิดจากเน็ตเวิร์กมีปัญหา ซึ่งทำให้ไคลเอนต์ ไม่สามารถรับ reply message ได้ในเวลากำหนดแล้วไคลเอนต์ต้องการยกเลิกการทำ transaction นั้นได้ แต่เพื่อความถูกต้องของข้อมูลทั้งสองฝั่งไคลเอนต์จะต้องสั่งให้เซิร์ฟเวอร์ทำการ rollback ข้อมูลในกรณีที่มีการ Update ข้อมูลลงบนไฟล์แล้ว หลังจากนั้นโปรแกรมไคลเอนต์จะหยุดการติดต่อกับโปรแกรม rpc ทางฝั่งเซิร์ฟเวอร์แต่เนื่องจากใช้การติดต่อแบบ Synchronous เมื่อฝั่งไคลเอนต์ส่ง request ไปก็จะทำการ block รอจนกว่าจะได้รับ reply message จากเซิร์ฟเวอร์ถึงจะทำงานอื่นต่อไปได้ ดังนั้นการที่จะยกเลิกการรอรับ reply message จากเซิร์ฟเวอร์นั้นจะต้องใช้ฟังก์ชั่น signal() เข้ามาช่วยในการอินเทอร์รัพ ซึ่งช่วงที่รอนั้นเมื่อมีการอินเทอร์รัพโดยการกดคีย์ Ctrl+C ก็จะไปทำฟังก์ชั่นที่สั่งให้เซิร์ฟเวอร์ทำการ rollback ข้อมูล ซึ่งมีการเรียกใช้ฟังก์ชั่นดังนี้



รูปที่ 3.26-1 การ interrupt signal สำหรับการทำ rollback

```
.....
signal(SIGINT,Roll_back);
.....
```

- SIGINT เป็นชนิดการ interrupt ต่อการกดคีย์ Ctrl+C
- Roll_back เป็นฟังก์ชั่นที่สั่งให้ Server ทำการ rollback ข้อมูล ที่ฝั่งเซิร์ฟเวอร์

วิธีการสั่งให้เซิร์ฟเวอร์ทำ rollback นั้นจะใช้วิธีของซ็อคเก็ตเข้ามาช่วยในการ implement การที่เลือกใช้ซ็อคเก็ตนี้เนื่องจากเร็วกว่า rpc ทำให้สามารถสั่งให้เซิร์ฟเวอร์ทำการ rollback ได้อย่างรวดเร็วกว่า โดยจะทำการเปิดซ็อคเก็ตทั้งสองฝั่งเพื่อทำการติดต่อ ดังนั้นทุกครั้งที่จะทำ transaction ใดๆก็จะนำค่าเดิมไปเก็บไว้ในไฟล์ temp ก่อนเพื่อไว้ใช้สำหรับกรณีที่จะทำ rollback ข้อมูล

จาก รูปที่ 3.2.6-1 แสดงขั้นตอนการทำ rollback เมื่อมีการเรียก interrupt signal ถ้าเกิดว่าเป็น Ctrl+c ก็ให้ไปทำส่วนที่สั่งให้ฟังก์ชัน rollback ฝั่งเซิร์ฟเวอร์ทำงานซึ่งจะใช้วิธีการของซ็อคเก็ตโดยโปรแกรมฝั่ง Server Socket จะเขียนแยกต่างหากจากโปรแกรม RPC ซึ่งจะเป็น deamon process ซึ่งรันตลอดเวลาเพื่อรอรับการเชื่อมต่อสำหรับทำการ rollback ข้อมูล

3.2.7 สัญญาณ (การอินเตอร์รัพโปรเซส)

สัญญาณ เป็นเหตุการณ์ที่สร้างขึ้นโดยระบบปฏิบัติการยูนิกซ์ ภายใต้เงื่อนไขที่กำหนดขึ้นบางอย่างเช่น เมื่อมีการละเมิดแอ็คเซสพื้นที่ในหน่วยความจำ เมื่อการคำนวณตัวเลขทศนิยมของซีพียู ไม่ถูกต้อง โดยโปรเซสสามารถส่งสัญญาณติดต่อกันได้ใน 2 กรณีคือ จากเคอร์เนลมายังโปรเซส หรือระหว่างโปรเซสด้วยกันเอง ส่วนการตอบสนองต่อสัญญาณที่ได้รับจะมีหลายอย่างด้วย เช่น ทำงานตามที่กำหนดไว้(Catch) หรือไม่สนใจต่อสัญญาณที่ได้รับ (ignore)

รายชื่อของสัญญาณที่กำหนดไว้ในไฟล์ signal.h มีดังนี้

*SIGABORT	จะสร้างขึ้นเมื่อเกิดข้อผิดพลาดจากการทำงานของฮาร์ดแวร์ และส่งไปยัง โปรเซสเอง เพื่อหยุดการทำงาน
SIGALRM	กำหนดเวลาของโปรเซส (ไม่ใช่เวลาที่ใช้งานซีพียู) ส่วนใหญ่จะใช้กำหนดเวลาหมดการทำงานของโปรเซส เช่น โปรแกรม ping
*SIGPIPE	จะขึ้นกับฮาร์ดแวร์ที่ใช้งานด้วย เช่น 4.3BSD บนเครื่อง VAX จะหมายถึงมีข้อผิดพลาดในการคำนวณจุดทศนิยมหรือการจำนวนตัวเลขที่เกิดขึ้น
SIGHUP	จะถูกส่งไปยังโปรเซสลูก เมื่อโปรเซสต้นฉบับสิ้นสุดการทำงาน
*SIGILL	ระบุว่ามีการใช้คำสั่งที่ไม่ถูกต้อง
SIGINT	จะเกิดขึ้นเมื่อกดคีย์ interrupt(Ctrl-C) บนเทอร์มินอล
SIGKILL	หยุดการทำงานของโปรเซส
SIGPIPE	เขียนข้อมูลไปยังไปป์
SIGQUIT	เกิดขึ้นเมื่อ กดคีย์ QUIT บนเทอร์มินอล

*SIGSEGV	เมื่อมีแอ็คเซสพื้นที่หน่วยความจำที่ไม่ถูกต้อง (จะขึ้นกับฮาร์ดแวร์ด้วย)
SIGTERM	เป็นสัญญาณส่งไปให้โปรแกรมเมื่อใช้คำสั่ง kill
SIGUSR1	เป็นสัญญาณที่ผู้ใช้งานกำหนดขึ้นนั้น(ซึ่งมีอยู่ 2 ชนิด) ใช้สำหรับติดต่อระหว่างโปรเซสเพื่อส่งข้อมูลบางอย่าง เช่น PID ของโปรเซสที่ส่งไป
SIGUSR2	เช่นเดียวกับ SIGUSR1 ข้างต้น ถ้าโปรเซสได้รับสัญญาณเหล่านี้ โดยไม่มีการเตรียมตัวจะทำให้โปรเซสหยุดทำงานในทันที สำหรับสัญญาณที่มีเครื่องหมาย * หมายถึง การทำงานต่างๆ จะขึ้นอยู่กับฮาร์ดแวร์ที่ใช้งานด้วย (System dependent)

นอกจากนี้ยังมีสัญญาณอื่นเพิ่มเติม ดังต่อไปนี้

SIGCHLD	จะส่งไปยังโปรเซสต้นฉบับเมื่อ โปรเซสลูกหยุดทำงาน หรือ พบคำสั่ง exit
SIGCONT	ถ้าโปรเซสหยุดทำงาน ให้เริ่มรันโปรเซสต่อไป
SIGSTOP	(สั่งให้โปรเซส)หยุดรัน(โปรเซสที่ได้รับสัญญาณไม่สามารถจะทำงานอื่นได้หรือเพิกเฉยต่อสัญญาณนี้ได้) โปรเซสที่ได้รับสัญญาณนี้สามารถรันต่อไปได้ด้วยสัญญาณ SIGCONT
SIGTSTP	จะส่งไปยังโปรเซสเมื่อกดคีย์ Ctrl-Z หรือ Ctrl-Y โปรเซสที่หยุดทำงานเนื่องจากสัญญาณนี้ สามารถทำงานได้ต่อด้วยสัญญาณ SIGCONT
SIGTTIN	จะส่งสัญญาณเมื่อแบ็คกราวน์โปรเซส จะอ่านข้อมูลจากเทอร์มินอล
SIGTTOU	จะส่งสัญญาณเมื่อแบ็คกราวน์โปรเซส จะเขียนข้อมูลจากเทอร์มินอล
SIGCHLD	โดยเดฟฟอลต์แล้ว จะถูกเพิกเฉย นอกนั้นจะทำให้โปรเซสที่รับสัญญาณหยุดทำงานยกเว้น SIGCONT จะทำให้โปรเซสทำงานต่อ สัญญาณเพิ่มเติมเหล่านี้จะใช้งานโดยเชลล์สคริปต์เป็นส่วนใหญ่ ถ้าเชลล์และเทอร์มินอล ได้รับการคอนฟิกให้ทำงานเป็นปกติแล้ว เมื่อกด Ctrl-C จะทำให้ระบบส่งสัญญาณ SIGINT มายังโปรเซสที่กำลังรันเป็น foreground process จะทำให้โปรแกรมหยุดทำงานทันที ถ้าไม่ได้กำหนดให้ทำงานอย่างอื่น (catch signal) สำหรับการตอบสนองเมื่อได้รับสัญญาณทำได้โดยใช้ไลบารี

ฟังก์ชั่น signal ต่อไปนี้

```
#include <signal.h>
void (*signal(int sig, void(*func)(int)))(int);
```

sig	จะระบุสัญญาณที่ได้รับ
void func	ฟังก์ชั่นที่จะทำงานเมื่อได้รับสัญญาณ(ไม่มีการส่งค่ากลับ)
int	ฟังก์ชันจะต้องมีอาร์กิวเมนต์สำหรับรับข้อมูล เป็นหมายเลขของสัญญาณ

ฟังก์ชั่น signal จะส่งค่ากลับเป็นค่าที่ได้กำหนดไว้ตั้งแต่การทำงานครั้งสุดท้าย หรือเป็นค่าใดค่าหนึ่งต่อไปนี้

SIG_IGN	เพิกเฉยต่อสัญญาณนั้น

SIG_DFL รีสโตร์การทำงานที่เป็นเดฟฟอลต์

ถ้าไม่กำหนดให้เพิกเฉยหรือให้ทำงานฟังก์ชั่นที่กำหนด จะทำงานที่เป็นเดฟฟอลต์โดยอัตโนมัติ

3.3 โครงสร้างข้อมูล(Data Structure)

โครงสร้างข้อมูลนี้ จะเป็นโครงสร้างที่เก็บข้อมูลเกี่ยวกับหมายเลขบัญชีและรหัสผ่านของผู้ใช้ที่จะทำการส่งไปยังฝั่งเซอร์เวอร์เพื่อที่จะทำการตรวจสอบ มีลักษณะเป็นดังนี้

```
struct usrinfo
        { string ID_account<15>;
          string Password<5>;
        };
```

- ID_account<15> : เป็นตัวแปรที่ใช้เก็บข้อมูลหมายเลขบัญชีของเจ้าของบัตร ATM ซึ่งกำหนดให้เป็นข้อมูลชนิดสตริงขนาด 15 ไบต์
- Password<5> : เป็นตัวแปรที่ใช้เก็บข้อมูลรหัสผ่านของผู้ใช้แต่ละคนที่เป็นเจ้าของบัตร

โครงสร้างนี้จะเป็นโครงสร้างที่อยู่ในไฟล์ดาต้าเบสที่เก็บข้อมูลเกี่ยวกับหมายเลขบัญชีและค่ายอดเงินของทุกๆผู้ใช้ จะใช้ตอนที่เราจะอ่านข้อมูลจากไฟล์ดาต้าเบส มีลักษณะดังนี้

```
Struct balance
        { string ID_account<15>;
          string  Firstname<20>;
          string  Lastname<30>;
          string  Balance<15>;
        };
```

- Firstname<20> : เป็นตัวแปรที่ใช้เก็บข้อมูลชื่อของผู้ใช้แต่ละคนเป็นข้อมูลชนิดสตริงขนาด 20 ไบต์
- Lastname<30> : เป็นตัวแปรที่ใช้เก็บข้อมูลนามสกุลของผู้ใช้แต่ละคนเป็นข้อมูลชนิดสตริงขนาด 30 ไบต์
- Balance<15> : เป็นตัวแปรที่ใช้เก็บข้อมูลยอดเงินล่าสุดของแต่ละผู้ใช้เป็นข้อมูลชนิดสตริงขนาด 15 ไบต์

โครงสร้างข้อมูลนี้จะเป็นโครงสร้างที่แสดงถึงผลลัพธ์ที่จะทำการส่งกลับมาจากเซิร์ฟเวอร์เพื่อให้ไคลเอนต์ทำไปแสดงบอกแก่ผู้ใช้ ภายในก็จะประกอบไปด้วย ยอดเงินคงเหลือ , หมายเลขบัญชีของผู้ใช้ , เวลาและวันที่เป็นต้น ดังนี้

```
Struct balance_resulttype
```

```
{ string Balance<15>;
  string ID_account<15>;
  string Date_Time<>;
  int    Flag;
};
```

- Balance<15> : เป็นตัวแปรที่ใช้เก็บข้อมูลยอดเงินที่ได้ทำการอัพเดทล่าสุดเป็นข้อมูลชนิดสตริงขนาด 15 ไบต์
- ID_account<15> : เป็นตัวแปรที่ใช้เก็บข้อมูลหมายเลขบัญชีของผู้ใช้เป็นข้อมูลชนิดสตริงขนาด 15 ไบต์
- Date_Time<> : เป็นตัวแปรที่ใช้เก็บข้อมูลเกี่ยวกับเวลาและวันที่เมื่อสิ้นสุดในการทำงานนั้นๆ
- Flag : เป็นตัวแปรที่ใช้เก็บข้อมูลแสดงว่ามีการสร้างไฟล์ล็อกไว้เพื่อบอกว่าขณะนั้นได้มีผู้ใช้กำลังทำการเปลี่ยนแปลงข้อมูลยอดเงินอยู่หรือไม่ จะมีค่าดังนี้

  0 หมายถึงว่าไม่มีการสร้างล็อกไฟล์สามารถเข้าใช้งานทำการเปลี่ยนแปลงค่ายอดเงินได้

  -1 หมายถึงว่ามีการสร้างล็อกไฟล์อยู่แล้วผู้ใช้คนใหม่ต้องรอก่อนจะเข้าไปเปลี่ยนแปลงยอดเงินได้

โครงสร้างนี้จะบอกเกี่ยวกับค่าที่จะส่งไปให้ทางฝั่งเซิร์ฟเวอร์ทำการถอนเงิน ก็จะประกอบไปด้วยหมายเลขบัญชีของผู้ใช้ และจำนวนเงินที่จะทำการถอน ซึ่งเซิร์ฟเวอร์ก็จะนำข้อมูลนี้ไปทำการหักออกจากยอดเงินในไฟล์ดาต้าเบส จะมีโครงสร้างดังนี้

```
struct withdraw_moneytype
{ string ID_account<15>;
  string Amount_money<15>;
};
```

- ID_account<15> : เป็นตัวแปรที่ใช้เก็บข้อมูลหมายเลขบัญชีของเจ้าของบัตร ATM ซึ่งกำหนดให้เป็นข้อมูลชนิดสตริงขนาด 15 ไบต์
- Amount_money<15> : เป็นตัวแปรที่ใช้เก็บข้อมูลจำนวนเงินที่จะทำการถอนเป็นตัวแปรชนิดสตริงขนาด 15 ไบต์

โครงสร้างนี้ก็เช่นเดียวกันจะบอกถึงค่าที่จะส่งไปยังฝั่งเซิร์ฟเวอร์เพื่อทำการโอนย้ายเงิน ก็จะประกอบไปด้วย หมายเลขบัญชีของผู้ใช้, หมายเลขบัญชีของผู้ที่จะได้รับโอน และจำนวนเงินที่จะทำการโอนจะมีโครงสร้างลักษณะเป็นดังนี้

```
Struct transfer_moneytype
{ string ID_account<15>;
  string Destination_id<15>;
```

```
    string Num_trans<15>;
};
```

- Destination_id<15> : เป็นตัวแปรที่ใช้เก็บข้อมูลหมายเลขบัญชีของผู้ที่ได้รับโอนเป็นตัวแปรชนิดสตริงขนาด 15 ไบต์
- Num_trans<15> : เป็นตัวแปรที่ใช้เก็บข้อมูลจำนวนเงินที่ต้องการโอนเป็นตัวแปรชนิดสตริงขนาด 15 ไบต์

โครงสร้างนี้เป็นข้อมูลที่จะบอกเกี่ยวกับผลลัพธ์สุดท้ายในการกระทำการโอนเงินเรียบร้อยแล้ว ซึ่งภายในจะประกอบไปด้วยฟิลด์ที่สำคัญดังนี้คือ หมายเลขบัญชีของผู้ใช้ , จำนวนยอดเงินคงเหลือ , หมายเลขบัญชีของผู้ได้รับโอน , จำนวนยอดเงินคงเหลือของผู้ได้รับโอน , วันที่และเวลาเป็นต้น

```
Struct transfer_resulttype
    { string ID_account<15>;
      string  Destination_id<15>;
      string  Num_trans<15>;
      string  Balance<15>;
      string  Firstname_des<20>;
      string  Lastname_des<30>;
      string  Date_Time<>;
      int     Flag_dest_id;
      int     Flag;
    };
```

- Firstname_des<20> : เป็นตัวแปรที่ใช้เก็บข้อมูลชื่อผู้ที่ได้รับโอนซึ่งเป็นตัวแปรชนิดสตริงขนาด 20 ไบต์
- Lastname_des<30> : เป็นตัวแปรที่ใช้เก็บข้อมูลนามสกุลของผู้ที่ได้รับโอนซึ่งเป็นตัวแปรชนิดสตริงขนาด 30 ไบต์
- Balance<15> : เป็นตัวแปรที่ใช้เก็บข้อมูลยอดเงินล่าสุดของผู้ที่ทำการโอนเป็นตัวแปรชนิดสตริงขนาด 15 ไบต์
- Num_trans<15> : เป็นตัวแปรที่ใช้เก็บข้อมูลจำนวนเงินที่ได้ทำการโอนเป็นตัวแปรชนิดสตริงขนาด 15 ไบต์
- Flag : เป็นตัวแปรที่ใช้เก็บข้อมูลแสดงว่ามีการสร้างไฟล์ล็อกไว้เพื่อบอกว่าขณะนั้นได้มีผู้ใช้กำลังทำการเปลี่ยนแปลงข้อมูลยอดเงินอยู่หรือไม่ จะมีค่าดังนี้

    0 หมายถึงว่าไม่มีการสร้างล็อกไฟล์สามารถเข้าใช้งานทำการเปลี่ยนแปลงค่ายอดเงินได้

-1 หมายถึงว่ามีการสร้างล็อกไฟล์อยู่แล้วผู้ใช้คนใหม่ต้องรอก่อนจะเข้าไปเปลี่ยนแปลงยอดเงินได้

ไฟล์ customer.dat ใช้สำหรับเก็บข้อมูลเกี่ยวกับชื่อหมายเลขบัญชี และรหัสผ่านของผู้ใช้งาน
ไฟล์ balance.dat ใช้สำหรับเก็บข้อมูลเกี่ยวกับหมายเลขบัญชี ชื่อ นามสกุลและยอดเงินคงเหลือของผู้ใช้งาน

โครงสร้างข้อมูลนี้จะต่างกับที่ผ่านมาคือจะมีลักษณะที่เรียกว่า union มีลักษณะข้อมูลเป็นเหมือนเคส โดยจะให้ case 0 หมายถึงกรณีมีข้อมูลส่งกลับ หรือเกิดมีข้อผิดพลาดที่ฝั่งเซิร์ฟเวอร์ก็จะไม่มีข้อมูลส่งกลับมา ไว้ตรวจสอบหมายเลขบัญชีของผู้ได้รับโอน

```
union transfer_resulttype switch(int errno)
 {
    case 0:  transfer_type transfer;
    default:  void;
 };
```

โครงสร้างนี้ก็เช่นเดียวกันกับโครงสร้างข้างต้น แต่จะเป็นการตรวจสอบข้อผิดพลาดของยอดเงินที่ส่งกลับมา

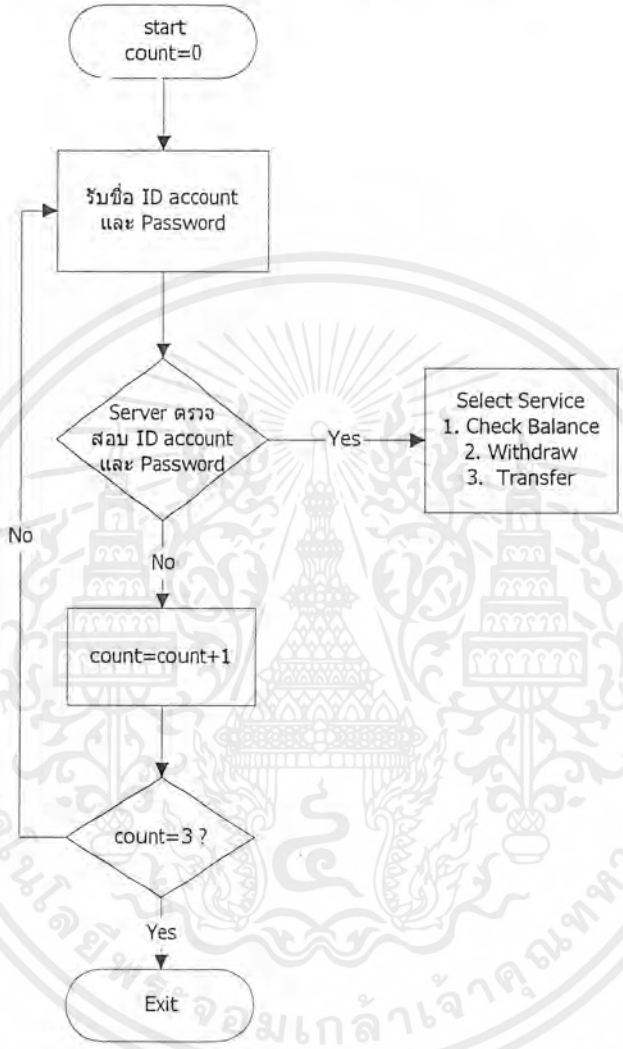
```
union balance_resulttype switch(int errno)
 {
    case 0:  balance_type balances;
    default: void;
 };
```

ส่วนข้างล่างนี้จะเป็นส่วนที่เป็นลักษณะเหมือนโพรซีเดอร์ในภาษา C เป็นส่วนที่นำไปใช้ร่วมกันทั้งทางฝั่งไคลเอนต์และเซิร์ฟเวอร์ โดยจะประกอบไปด้วยหมายเลขของโปรแกรม หมายเลขของเวอร์ชั่นและหมายเลขของโพรซีเดอร์แต่ละโพรซีเดอร์ ดังนี้เป็นต้น

```
program ATMPROG {
        version ATMVERS {
                usrinfo VALIDATE(usrinfo)=1;
                balance_resulttype CHECK_BALANCE(string) = 2;
                balance_resulttype WITHDRAW(withdraw_moneytype) = 3;
                transfer_resulttype TRANSFER(transfer_moneytype) = 4;
                }=1;
        }=123456788;
```

บทที่ 4
อัลกอริทึม

4.1 การส่งหมายเลขบัญชีและรหัสผ่านสำหรับตรวจสอบเพื่อขอใช้บริการ



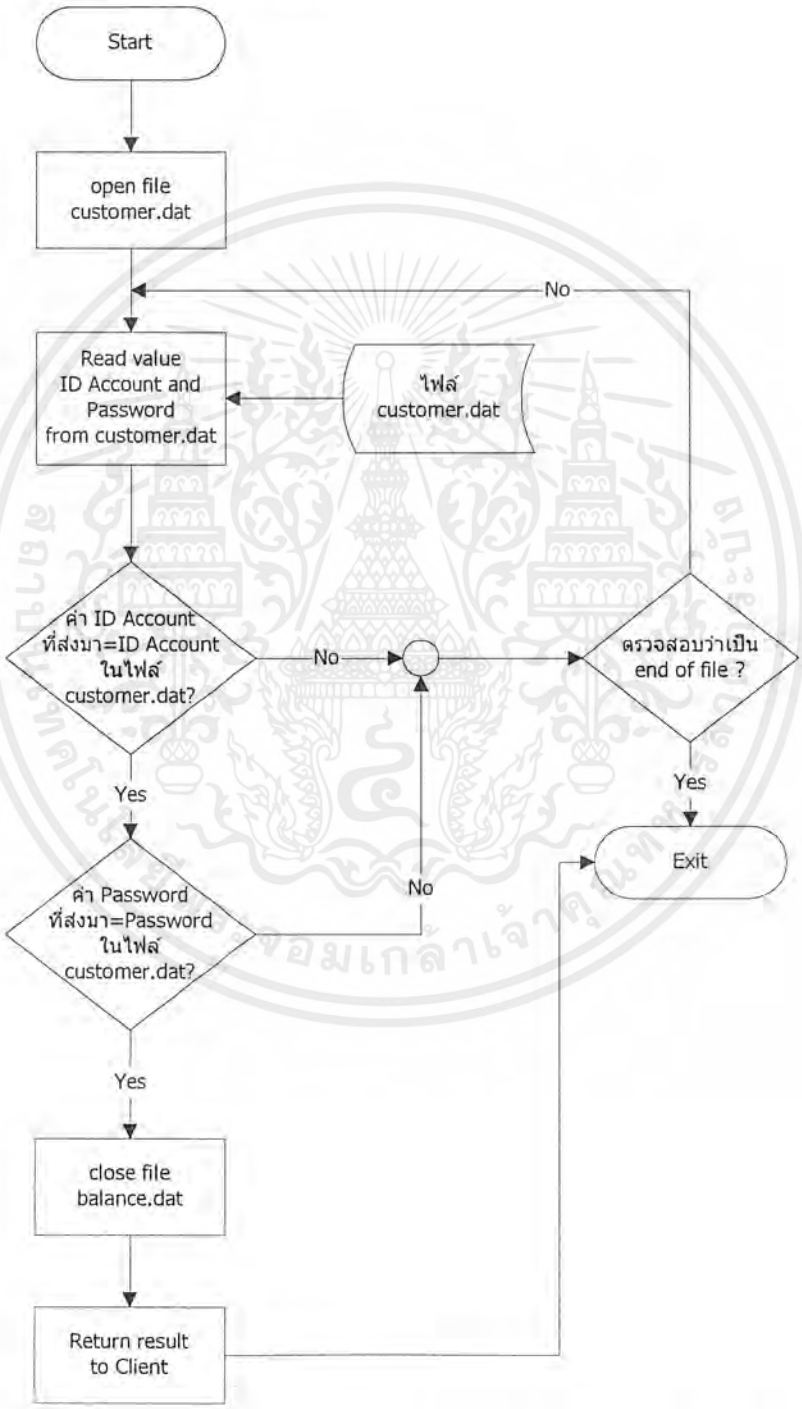
รูปที่ 4.1-1 การทำงานของโพรซีเดอร์ส่งหมายเลขบัญชีและรหัสผ่าน

1. โปรแกรมจะทำการรับชื่อบัญชีของผู้ใช้และรหัสผ่านของผู้ใช้ โดยเริ่มต้นจะทำการกำหนดค่าเริ่มต้นของโปรแกรมให้ตัวแปร count = 0
2. ขั้นต่อไปโปรแกรมที่ฝั่งไคลเอนต์จะทำการส่งอาร์กิวเมนต์ที่เป็นเลขบัญชีและรหัสผ่านของผู้ใช้ไปยังฝั่งเซิร์ฟเวอร์
3. โปรแกรมทางฝั่งเซิร์ฟเวอร์จะทำการตรวจสอบเลขบัญชีและรหัสผ่านของผู้ใช้ที่ส่งมาจากฝั่งไคลเอนต์ แล้วทำการส่งผลลัพธ์กลับไปยังฝั่งไคลเอนต์
4. ค่าที่ส่งกลับมายังฝั่งไคลเอนต์ถ้ามีค่าเท่ากันกับค่าที่ส่งออกไปแสดงว่าเลขบัญชีและรหัสผ่านถูกต้อง

จะกระทำลำดับขั้นตอนต่อไป คือเลือกบริการที่จะทำ

5. แต่ถ้าค่าที่ส่งกลับมานั้นไม่เท่ากันกับค่าที่ส่งออกไป โปรแกรมจะทำการเพิ่มตัวแปร count ขึ้นอีกหนึ่ง และจะทำการรับเลขบัญชีและรหัสผ่านอีกครั้ง โดยที่ถ้าตัวแปร count มีค่าเท่ากับ 3 ครั้ง จะทำการออกจากโปรแกรมเพราะถือว่าทำการป้อนผิด 3 ครั้ง

4.2 การตรวจสอบหมายเลขบัญชีและรหัสผ่าน

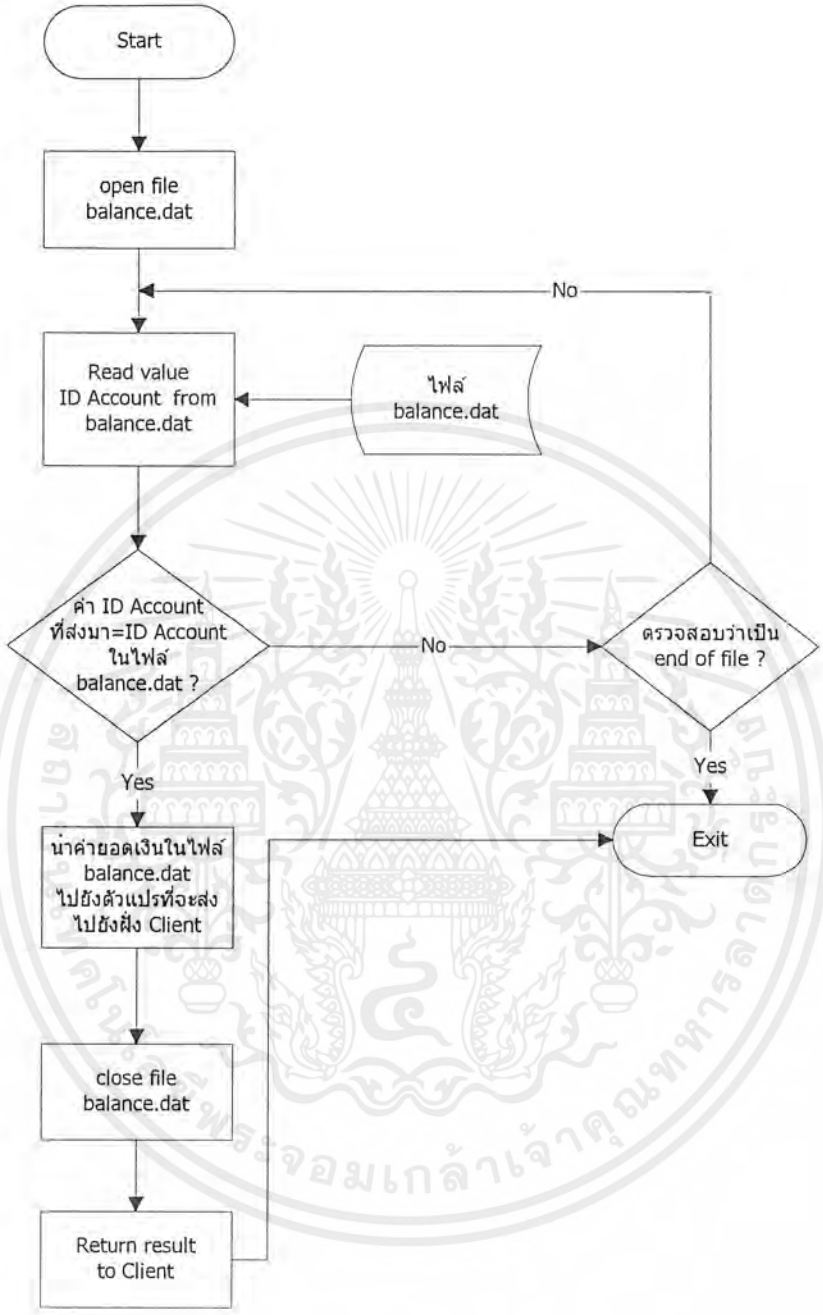


รูปที่ 4.2-1 การทำงานของโพรซีเดอร์ ตรวจสอบหมายเลขบัญชีและรหัสผ่าน

1. โพรซีเดอร์ตรวจสอบเลขบัญชีและรหัสผ่านจะทำการรับอาร์กิวเมนต์ที่เป็นเรคคอร์ดที่ส่งมาจากฝั่งไคลเอนต์ ซึ่งบรรจุด้วยสองฟิลด์ คือ เลขบัญชี และรหัสผ่านของผู้ใช้
2. จากนั้นทำการจองเมมโมรีสำหรับตัวแปรต่างๆที่ต้องใช้งาน แล้วทำการเปิดไฟล์ customer.dat
3. ทำการอ่านค่าเลขบัญชีและรหัสผ่านของผู้ใช้จากไฟล์ที่เก็บเป็นดาต้าเบสขึ้นมาไว้ในตัวแปร
4. ต่อมาโพรซีเดอร์จะทำการตรวจสอบเลขบัญชีที่ได้จากไฟล์ดาต้าเบสกับเลขบัญชีที่ผู้ใช้ส่งมาจากฝั่งเซิร์ฟเวอร์ว่าเท่ากันหรือไม่ ถ้าเท่ากันก็จะไปทำในขั้นตอนต่อไป แต่ถ้าไม่เท่ากันก็จะทำการตรวจสอบว่าใช่ end of file หรือไม่ ถ้ายังก็จะทำการวนไปอ่านค่าเลขบัญชีและรหัสผ่านของเรคคอร์ดต่อไปในไฟล์ดาต้าเบสอีกครั้ง แล้วนำมาเปรียบเทียบอีกครั้งหนึ่ง
5. ถ้าเลขบัญชีที่ตรวจสอบนั้นตรงกัน ก็จะทำการตรวจสอบรหัสผ่านของผู้ใช้ว่าเท่ากันกับที่ส่งมาหรือไม่ ถ้าไม่เท่ากันก็จะทำเช่นเดียวกันกับการตรวจสอบเลขบัญชีเมื่อทำการตรวจสอบทั้งสองฟิลด์ว่าเท่ากันกับในไฟล์ดาต้าเบสแล้วก็จะส่งค่าผลลัพธ์กลับไปยังฝั่งไคลเอนต์

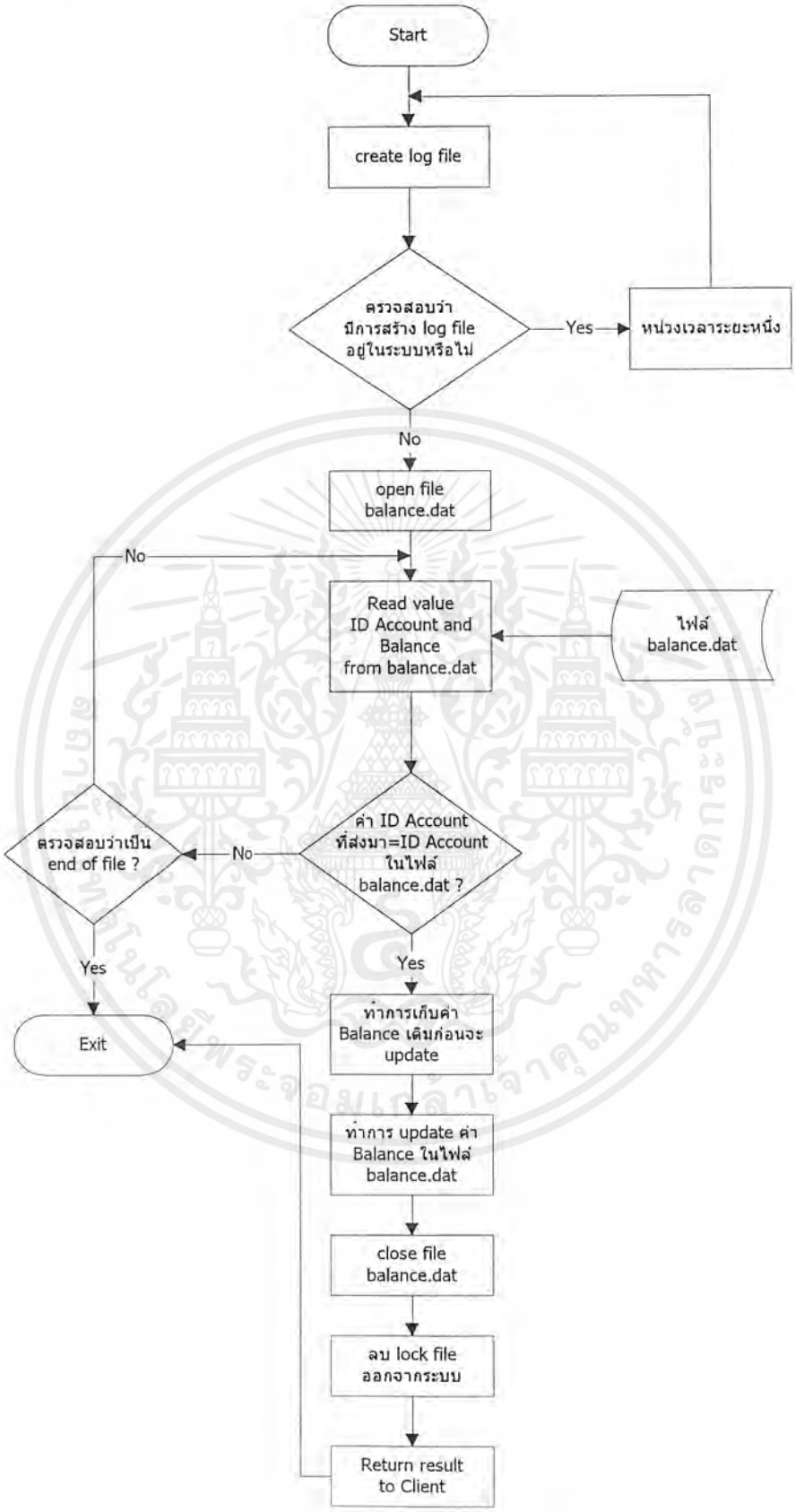
4.3 การทำขอดูยอดคงเหลือ

1. โพรซีเดอร์ดูค่ายอดเงิน (Check balance) จะทำการรับอาร์กิวเมนต์ที่เป็นหมายเลขบัญชีของผู้ใช้ ที่ส่งมาจากฝั่งไคลเอนต์
2. จากนั้นจะทำการเปิดไฟล์ดาต้าเบสชื่อ balance.dat ซึ่งเก็บค่าหมายเลขบัญชี , ชื่อ , นามสกุล และยอดเงินของผู้ใช้ทุกคน
3. ทำการอ่านค่าเลขบัญชีจากไฟล์ดาต้าเบสมาเก็บไว้ที่ตัวแปรหนึ่งที่ได้ทำการกำหนดไว้
4. ต่อมาโพรซีเดอร์จะทำการตรวจสอบหมายเลขบัญชีจากไฟล์ดาต้าเบสกับหมายเลขบัญชีที่ส่งมาจากฝั่งไคลเอนต์ว่าเท่ากันหรือไม่ ถ้าไม่เท่ากันจะตรวจสอบว่าใช่ end of file หรือไม่ ถ้าเท่ากับ end of file แล้วจะออกจากโปรแกรม แต่ถ้าไม่เท่ากับ end of file จะทำการตรวจสอบหมายเลขบัญชีที่ส่งมากับหมายเลขบัญชีเรคคอร์ดต่อไปในไฟล์ดาต้าเบสอีกครั้งหนึ่ง
5. ถ้าตรวจสอบแล้วหมายเลขบัญชีเท่ากันจะทำการนำค่ายอดเงินที่ได้อ่านขึ้นมาเก็บไว้ในตัวแปรหนึ่งมาใส่ยังตัวแปรที่จะทำการส่งกลับไปยังฝั่งไคลเอนต์ จากนั้นก็ทำการปิดไฟล์ balance.dat และทำการส่งค่ากลับไปยังฝั่งไคลเอนต์



รูปที่ 4.3-1 การทำงานของโพรซีเดอร์ขอดูยอดคงเหลือ

4.4 การถอนเงิน



รูปที่ 4.4-1 ทำงานของโพรซีเดอร์การถอนเงิน

1. โพรซีเดอร์ถอนเงิน (Withdraw balance) จะรับอาร์กิวเมนต์ที่เป็นสตรัคเจอร์ที่ส่งมาจากไคลเอนต์ ซึ่งภายในสตรัคเจอร์นี้จะประกอบไปด้วย 2 ฟิลด์ คือฟิลด์ที่เป็นหมายเลขบัญชีของผู้ที่ทำการถอนเงิน และฟิลด์ที่เป็นค่าของเงินที่จะทำการถอน

2. เมื่อรับเข้ามาแล้วทางฝั่งเซิร์ฟเวอร์จะทำการสร้างไฟล์ๆหนึ่งที่จะใช้เป็นตัวบอกว่ามีไคลเอนต์ตัวอื่นกำลังเข้ามาใช้งานอัพเดทข้อมูลในไฟล์คาต้าเบสหรือไม่ โดยจะมีค่าที่ส่งกลับมาบอกดังนี้

-1 แสดงว่ามีโปรเซสหรือไคลเอนต์ตัวหนึ่งกำลังทำการอัพเดทไฟล์คาต้าเบสนี้อยู่

0 แสดงว่าไม่มีโปรเซสใดกำลังอัพเดทข้อมูลอยู่ โปรเซสที่เข้ามาสามารถเริ่มการทำงานได้

ถ้าค่าที่ส่งกลับมาเป็น –1 จะทำการวนลูปเพื่อไปพยายามสร้างไฟล์ตัวนี้ใหม่อีกครั้ง โดยก่อนที่จะวนลูปไปนี้ได้ทำการหน่วงการทำงานไว้ช่วงเวลาหนึ่ง

3. เมื่อเข้ามาทำงานได้แล้ว จะทำการเปิดไฟล์ balance.dat ขึ้นมาเพื่อที่จะทำการอัพเดทข้อมูล

4. จากนั้นจะทำการอ่านค่าของบัญชีของผู้ใช้จะไฟล์ balance.dat เพื่อที่จะมาเปรียบเทียบกับฟิลด์ที่ส่งมาในสตรัคเจอร์ที่ส่งมานั้นว่าตรงกันหรือไม่ ถ้าตรงกันจะทำงานขั้นตอนต่อไป แต่ถ้าไม่ตรงกันจะตรวจสอบว่าใช่ end of file หรือไม่ ถ้าไม่ใช่ก็จะไปทำการอ่านค่าตัวต่อไปในไฟล์มาเปรียบเทียบแต่ถ้าเป็น end of file แล้วจะจบการทำงาน ไม่สามารถหาข้อมูลมาทำการอัพเดทได้

5. เมื่อเลขบัญชีตรงกันจะมาทำการเก็บค่าก่อนจะทำการอัพเดทไว้ในไฟล์ๆหนึ่ง เพื่อนำกลับมาใช้อัพเดทค่าให้เป็นค่าเดิม เนื่องจากกรณีการติดต่อสื่อสารไม่สำเร็จ อาจจะเนื่องมาจากไคลเอนต์หรือเซิร์ฟเวอร์เกิดมีปัญหาบางอย่างขึ้นมา

6. จากนั้นก็จะทำการอัพเดทค่าในไฟล์คาต้าเบสนั้น และทำการปิดไฟล์

7. ต่อมาก็จะทำการลบไฟล์ที่สร้างขึ้นเพื่อให้ไคลเอนต์นั้นๆทำการอัพเดทไฟล์ได้ เพื่อจะได้ให้ไคลเอนต์ตัวอื่นได้เข้ามาทำการอัพเดทไฟล์ได้

8. สุดท้ายจะส่งค่ายอดเงินคงเหลือกลับไปแสดงที่ฝั่งไคลเอนต์

4.5 การโอนเงิน

1. โพรซีเดอร์โอนเงินจะเป็นการอัพเดทข้อมูลระหว่างผู้โอนและผู้ได้รับโอน โดยทางฝั่งเซิร์ฟเวอร์จะทำการรับข้อมูลซึ่งเป็นสตรัคเจอร์ของการโอนเงินมาจากฝั่งไคลเอนต์โดยที่ภายในจะประกอบด้วยฟิลด์หลักๆ เช่นหมายเลขบัญชีของผู้โอน , หมายเลขบัญชีของผู้ได้รับโอน และจำนวนเงินที่จะทำการโอน

2. ขั้นต่อมาโพรซีเดอร์นี้จะทำการสร้าง lock file เพื่อที่จะทำการให้ผู้ใช้สามารถที่จะทำการอัพเดทข้อมูลและเพื่อป้องกันไม่ให้ผู้อื่นมาทำการอัพเดทข้อมูลของเรา จะทำให้เกิดการผิดพลาดได้ โดยจะส่งค่ากลับมาว่ามีไฟล์นี้อยู่ในระบบหรือไม่โดยจะเป็นค่าดังนี้

-1 แสดงว่ามีโปรเซสหรือไคลเอนต์ตัวหนึ่งกำลังทำการอัพเดทไฟล์คาต้าเบสนี้อยู่

0 แสดงว่าไม่มีโปรเซสใดกำลังอัพเดทข้อมูลอยู่ โปรเซสที่เข้ามาสามารถเริ่มการทำงานได้

ถ้าค่าที่ส่งกลับมาเป็น –1 จะทำการวนลูปเพื่อไปพยายามสร้างไฟล์ตัวนี้ใหม่อีกครั้ง โดยก่อนที่จะวนลูปไปนี้ได้ทำการหน่วงการทำงานไว้ช่วงเวลาหนึ่ง

3. ขั้นตอนต่อมาจะทำการตรวจสอบหมายเลขของผู้โอนว่ามีอยู่ในไฟล์คาต้าเบสหรือไม่ ถ้ามีก็จะเริ่มทำ

การโอนเงิน แต่ถ้าไม่มีก็จะทำการแจ้งไปบอกทางฝั่งไคลเอนต์ว่าหมายเลขของผู้โอนผิด จะออกจากการทำงานทันที

4. ทำการเปิดไฟล์ที่เก็บยอดจำนวนเงินขึ้นมา จากนั้นจะทำการนำเอาฟิลด์ที่เป็นหมายเลขบัญชีของผู้โอนมาเปรียบเทียบกับที่อ่านขึ้นมาได้จากไฟล์ดาต้าเบสว่าตรงกันหรือไม่ ถ้าไม่ตรงกันก็จะทำการวนลูปไปอ่านค่าหมายเลขบัญชีตัวต่อไปในไฟล์ดาต้าเบสมาทำการตรวจสอบอีก จนกว่าจะเจอหรือถึงจุด end of file ถ้าเจอจุดนี้แล้วยังไม่เจอจะออกจากโปรแกรม

5. จากนั้นจะทำการเก็บค่ายอดเงินเดิมของผู้ที่จะทำการโอนไว้ เพราะว่าอาจจะเกิดเหตุการณ์ทำให้การสื่อสารไม่ประสบความสำเร็จได้ ซึ่งอาจจะเกิดได้จากทั้งฝั่งเซิร์ฟเวอร์และไคลเอนต์ โดยจะเก็บไว้ที่ไฟล์ๆ หนึ่ง

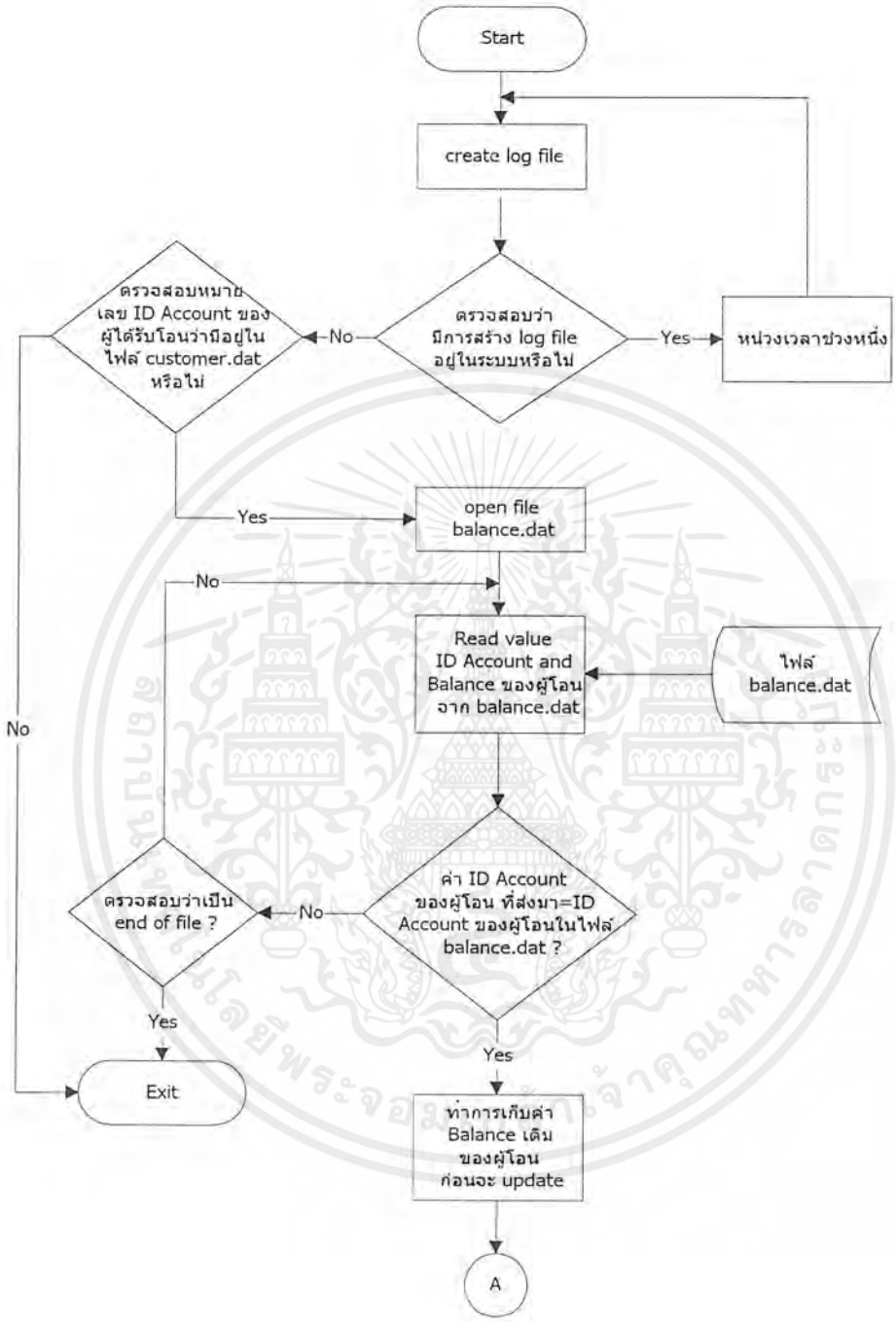
6. ขั้นต่อมาจะทำการจะทำการอัพเดทยอดเงินของผู้ที่จะทำการโอน

7. จะทำการตรวจเช็คว่าค่าหมายเลขบัญชีของผู้โอนที่อยู่ในฟิลด์ที่ทำการส่งมาจากฝั่งไคลเอนต์นั้นเท่ากันกับค่าหมายเลขบัญชีที่ได้มาจากไฟล์ดาต้าเบสหรือไม่ ก็จะทำเช่นเดียวกับที่ผ่านมา

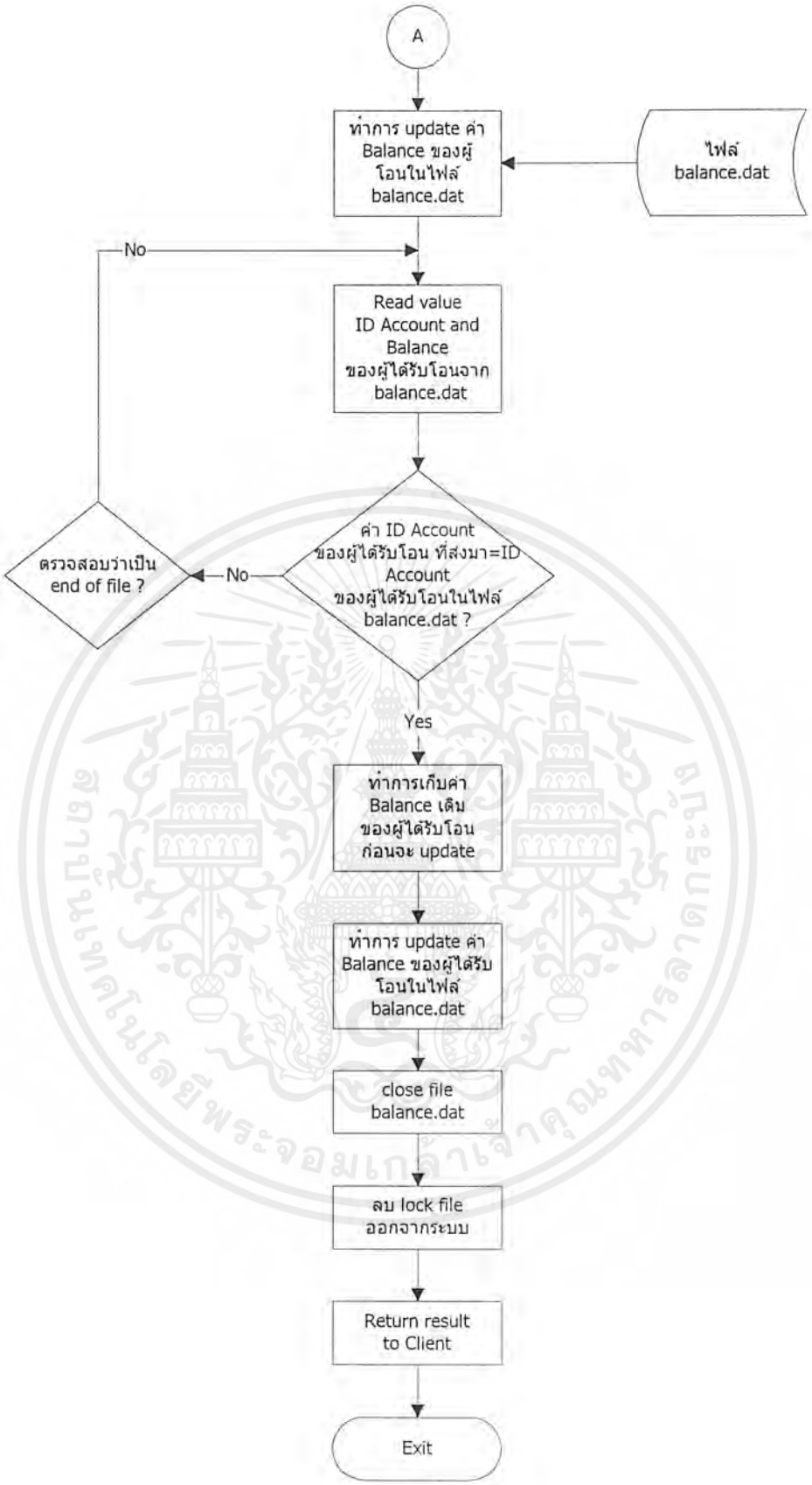
8. จากนั้นก็จะทำการเก็บค่ายอดเงินเดิมของผู้ที่ได้รับการโอนไว้ในไฟล์อีกไฟล์หนึ่ง และเริ่มทำการอัพเดทค่ายอดเงินของผู้ที่ได้รับการโอน

9. ทำการปิดไฟล์และลบ lock file ที่สร้างไว้เพื่อที่จะให้ไคลเอนต์ตัวอื่นได้เข้ามาทำการอัพเดทไฟล์ได้

10. สุดท้ายจะส่งค่ายอดเงินคงเหลือของผู้ที่ทำการโอนกลับไปยังฝั่งไคลเอนต์



รูปที่ 4-5 การทำงานของโพรซีเดอร์โอนเงิน(1)



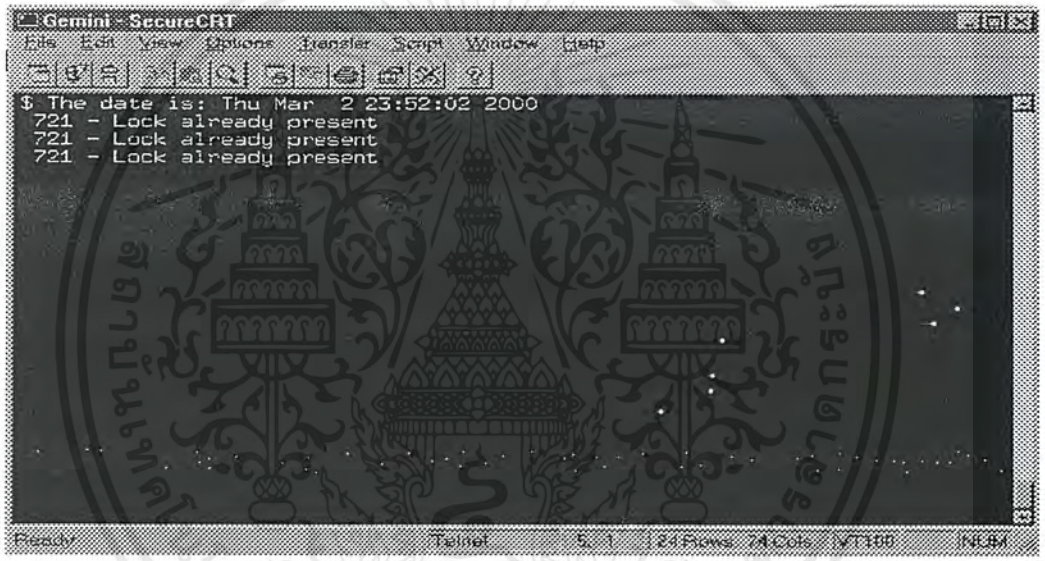
รูปที่ 4.5-2 การทำงานของโพรซีเดอร์โอนเงิน(2)

บทที่ 5
การทดสอบและผลการทดสอบ

5.1 ทดสอบการทำ lockfile ของโปรแกรม

โปรแกรมจะเรียกฟังก์ชัน open เพื่อสร้างไฟล์ lockfile โดยใช้แฟลก O_CREAT และ O_EXCL เพื่อสร้างไฟล์ขึ้นมาใหม่ จะใช้ตัวแปร fildes เป็นตัวรับค่าที่ส่งมาจากการเปิดไฟล์ โดยมีค่าดังนี้

- 1 หมายถึงว่ามีไฟล์นั้นอยู่ในระบบแล้วก็ส่งค่ากลับไปยังฝั่งไคลเอนต์ให้ทำการรอจนกว่าโปรเซสที่กำลังใช้งานจะใช้งานไฟล์นั้นเสร็จสิ้น
- 0 หมายถึงว่าไม่มีไฟล์นั้นอยู่ในระบบและทำการสร้างไฟล์ lockfile ขึ้นมาเพื่อบอกว่าไคลเอนต์ตัวนี้ได้ทำการเปลี่ยนแปลงข้อมูลในไฟล์นั้นอยู่



รูปที่ 5.1-1 ลักษณะของโปรเซสที่กำลังอัพเดทไฟล์อยู่

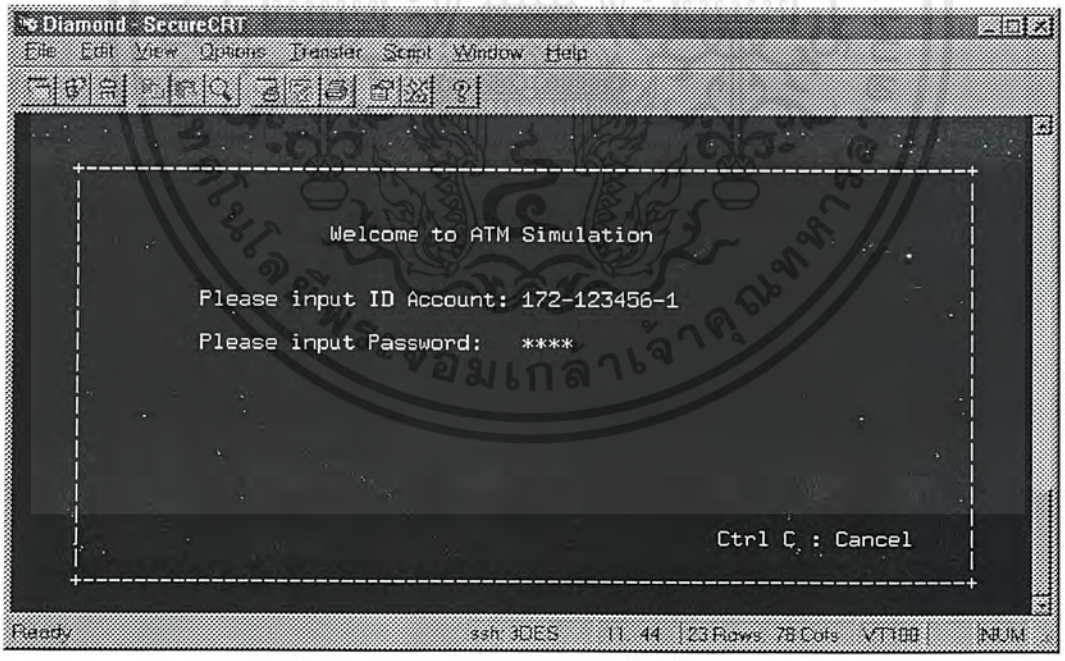
จากรูปที่ 5.1-1 มีโปรเซสหรือไคลเอนต์ตัวหนึ่งกำลังทำการอัพเดทค่าในไฟล์ดาต้าเบส ทางฝั่งเซอร์เวอร์จะแสดงแมสเซสที่ว่ามีไคลเอนต์ทำงานอยู่ ไคลเอนต์ตัวอื่นที่เข้ามาใช้งานจะต้องทำการรอจนกระทั่งไคลเอนต์ตัวก่อนทำงานเสร็จเรียบร้อย

จากรูปที่ 5.1-2 จะเป็นการเข้าใช้งานอัพเดทไฟล์ดาต้าเบส เช่นการถอนเงินหรือโอนเงิน ไคลเอนต์จะสามารถทำงานได้เลย เนื่องจากไม่มีโปรเซสหรือไคลเอนต์ใดกำลังใช้งานอยู่

รูปที่ 5.1-2 การเข้าใช้งานอัพเดทไฟล์โดยไคลเอนต์

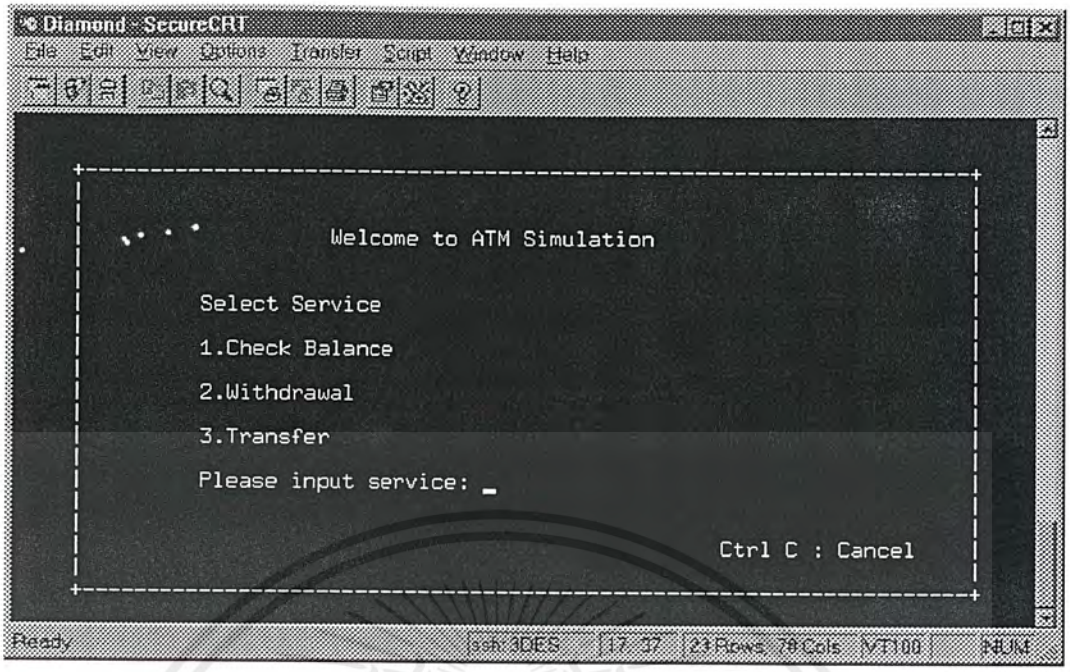
5.2 ผลหลังจากที่เข้าสู่โปรแกรม ATM Simulation

หลังจากที่สั่งให้โปรแกรมเซิร์ฟเวอร์ทำงานเป็น Deamon แล้ว ก็สั่งให้โปรแกรมไคล์เอ็นต์ทำงาน จากนั้นก็ใส่ ID และรหัสผ่านเพื่อขอใช้สิทธิ์ ซึ่งแสดงดังรูปที่ 5.2-1 ดังนี้



รูปที่ 5.2-1 หน้าจอแสดงเมื่อใส่ ID และรหัสผ่าน

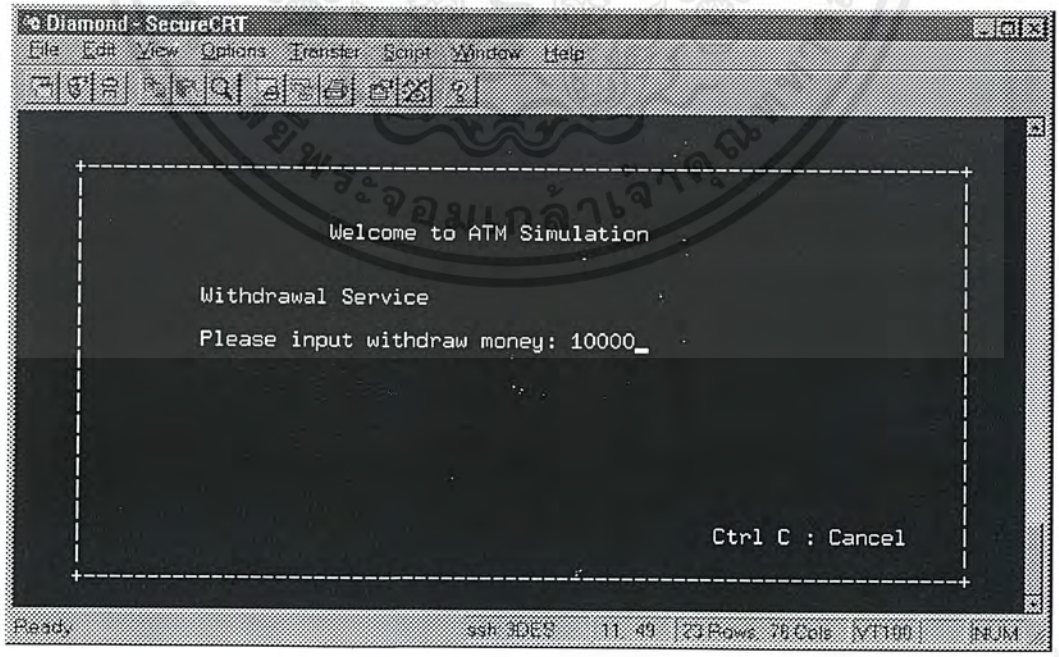
หลังจากเมื่อได้รับสิทธิ์ให้ใช้งานเข้าไป สามารถที่จะเลือกขอใช้บริการได้ซึ่งแสดงดังรูปที่ 5.2-2



รูปที่ 5.2-2 หน้าจอแสดงรายการต่างๆที่สามารถใช้บริการได้

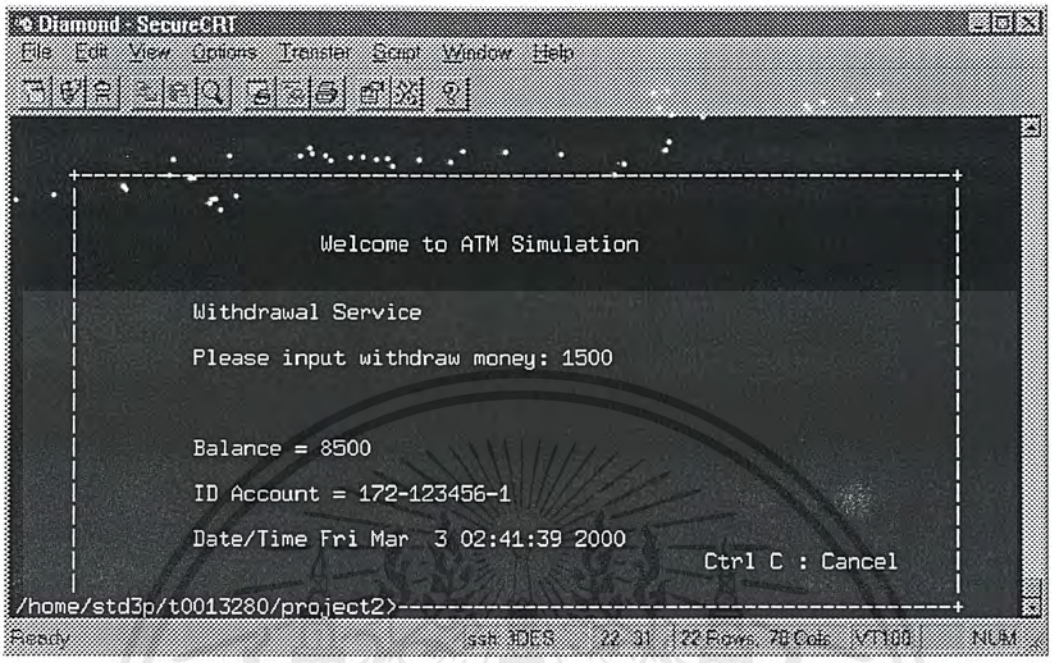
5.3 ผลการทำงานของโปรแกรมเมื่อมีการเรียกใช้บริการ การถอนเงิน

ถ้าเลือกบริการการถอนเงินก็ให้กดคีย์ 2 เพื่อเลือกบริการการถอนเงิน โดยจะต้องใส่จำนวนเงินไปด้วยซึ่งแสดงดังรูปที่ 5.3-1 หลังนั้นหมายเลข ID และ จำนวนเงินที่ต้องการถอนจะไปยังฝั่งเซิร์ฟเวอร์



รูปที่ 5.3-1 หน้าจอแสดงเมื่อเลือกบริการการถอนเงิน

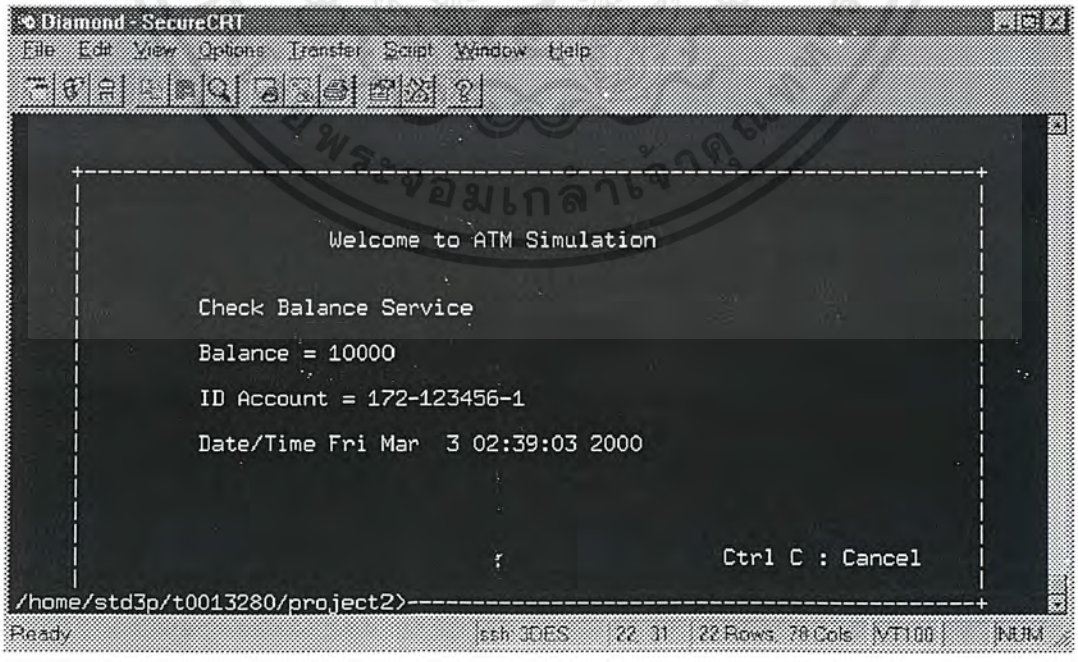
หลังจากเมื่อฝั่งเซิร์ฟเวอร์ประมวลเสร็จแล้ว ก็จะส่งผลลัพธ์กลับมายังฝั่งไคล์เอ็นต์ซึ่งผลที่ได้แสดงดังรูปที่ 5.3-2



รูปที่ 5.3-2 ผลลัพธ์หลังจากที่เซิร์ฟเวอร์ประมวลผลเสร็จแล้วส่งมายังฝั่งไคล์เอ็นต์

5.4 ผลการทำงานของโปรแกรมเมื่อมีการเรียกใช้บริการ ขอตรวจสอบยอดเงินคงเหลือ

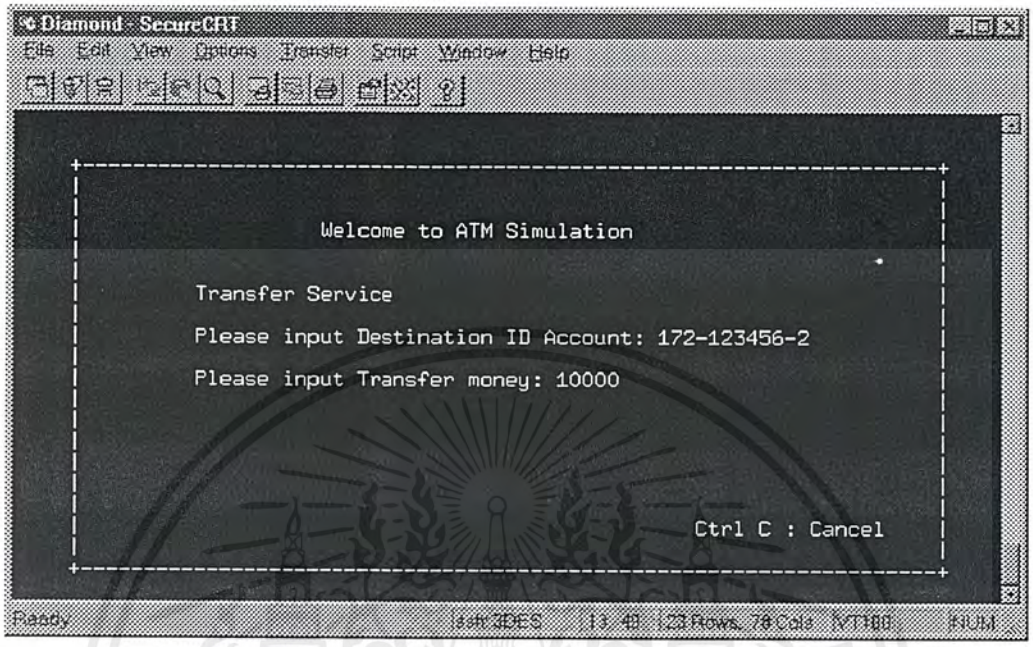
สำหรับบริการขอดูยอดคงเหลือทำได้โดยเลือกบริการด้วยเลข 1 ผลลัพธ์ที่ได้แสดงดังรูปที่ 5.4-1



รูปที่ 5.4-1 ผลลัพธ์ที่ได้จากโพรซีเดอร์ขอดูยอดคงเหลือ

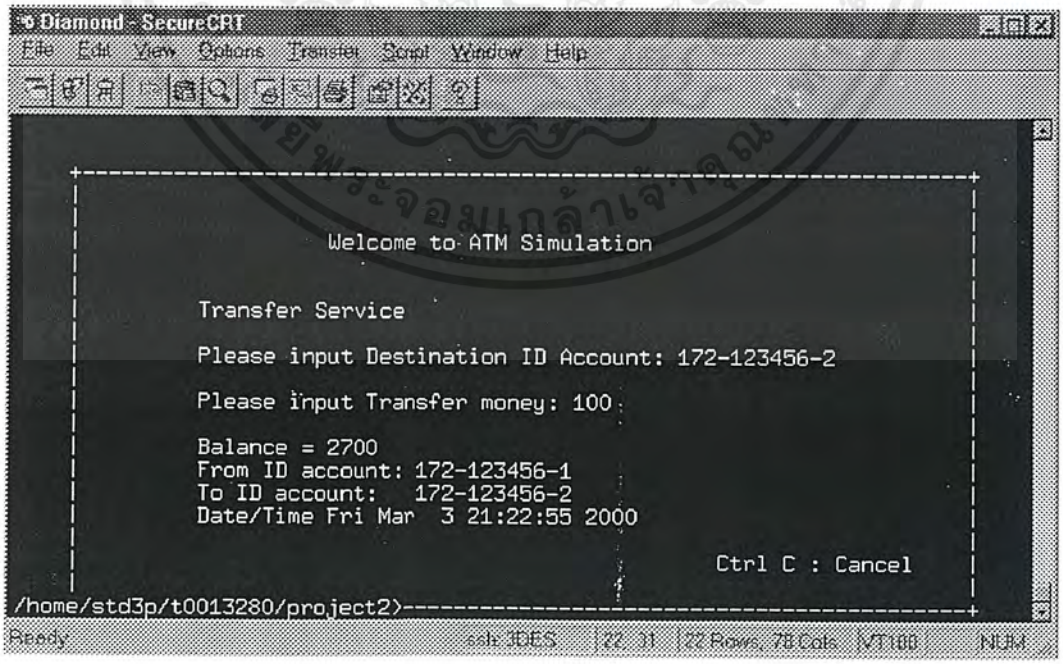
5.5 ผลการทำงานของโปรแกรมเมื่อมีการเรียกใช้บริการ การโอนเงิน

ในการขอใช้บริการการโอนเงินต้องมีการใส่ ID ที่ต้องการโอนไปพร้อมด้วยจำนวนเงินซึ่งแสดงดังรูปที่ 5.5-1



รูปที่ 5.5-1 หน้าจอแสดงการขอใช้บริการโอนเงิน

เมื่อเซิร์ฟเวอร์ประมวลผลเสร็จแล้วจะส่งผลกลับมายังไคล์เอ็นต์แสดงดังรูปที่ 5.5-2



รูปที่ 5.5-2 ผลลัพธ์จากการทำงานของโพรซีเดอร์โอนเงินแสดงที่ฝั่งไคลเอนต์

บทที่ 6
สรุปและบทวิจารณ์

สรุป

การพัฒนาระบบจำลองการทำงานของ ATM นี้ เป็นการให้บริการอย่างง่ายๆ 3 รูปแบบคือ การตรวจสอบยอดเงิน , การถอนเงินและการโอนเงิน ซึ่งลักษณะการทำงานของระบบเป็นแบบตอบรับกับผู้ใช้ทันทีหรือเรียกว่าระบบการทำงานแบบเรียลไทม์ โดยที่การทำงานประมวลผลทุกอย่างจะอยู่ที่ตัวเซิร์ฟเวอร์ ส่วนทางฝั่งไคลเอนต์จะทำหน้าที่เพียงรับอินพุตที่เข้ามาและแสดงเอาท์พุทที่ได้รับมาจากฝั่งเซิร์ฟเวอร์ ซึ่งข้อมูลต่างๆที่รับเข้ามาจะถูกส่งไปยังเซิร์ฟเวอร์ด้วยกลไกการทำงานของ RPC ทางฝั่งเซิร์ฟเวอร์นั้นจะมีการควบคุมการใช้ทรัพยากรหรือรีซอร์สของแต่ละโปรเซสเพื่อไม่ให้เกิดการผิดพลาดในการทำงานเมื่อมีการร้องขอเข้ามาจากหลายๆไคลเอนต์หรือมีผู้ใช้บริการเข้ามาเป็นจำนวนมาก อีกทั้งยังมีการควบคุมการทำงานเมื่อเกิดความผิดพลาดในการสื่อสารหรือความผิดพลาดที่เกิดจากระบบเน็ตเวิร์กที่ใช้เพื่อควบคุมความถูกต้องของข้อมูลในการอัพเดท

บทวิจารณ์

การทำงานของแอพพลิเคชั่นที่ทำงานบนระบบเครือข่ายมีความยุ่งยากซับซ้อนกว่าการทำงานบนระบบเดียวกันมาก ซึ่งจะต้องควบคุมการทำงานของทั้งสองฝั่งให้มีความ Synchronize กัน และในแอพพลิเค
ชั่นพวกนี้อาจจะเกิดปัญหาขึ้นได้มากกว่าการทำงานแบบระบบเดียวกัน ในระบบการจำลองการทำงานของ ATM นี้อาจจะไม่มีการทำงานที่เหมือนกับระบบจริงทุกประการก็เพราะว่าในการทำโครงงานนี้มีวัตถุประสงค์เพื่อให้เข้าใจในการทำงานของระบบไคลเอนต์เซิร์ฟเวอร์ที่ทำงานด้วยกลไกของ RPC การทำงานการประมวลผลในระบบเครือข่ายหรือที่เรียกว่าการประมวลผลแบบกระจาย และควบคุมการทำงานของโปรเซสจะบอกได้ว่าคือสร้างแอพพลิเคชั่นเพื่อทดสอบการทำงานของ RPC มากกว่าที่จะทำแอพพลิเคชั่นด้วย RPC

ข้อเสนอแนะและแนวทางการพัฒนาต่อ

ระบบจำลองนี้เป็นระบบที่ยังไม่สมบูรณ์เทียบเท่าการทำงานจริงๆของตู้ ATM ยังสามารถเพิ่มการทำงานต่างๆเข้าไปได้อีก ทั้งยังเรื่องเกี่ยวกับกลไกการทำงานของ RPC ซึ่งยังมีการทำงานอีกมากที่ยังไม่ได้ศึกษาเชื่อว่าถ้าได้ศึกษาการทำงานทั้งหมดของกลไกนี้จะทำให้การพัฒนาแอพพลิเคชั่นบนระบบเครือข่ายมีความสามารถในการทำงานมากยิ่งขึ้น ทางด้านระบบของ user interface ยังเป็นระบบเท็กซ์โหมด ซึ่งอาจจะพัฒนาให้เป็นระบบกราฟิกส์เพื่อการแสดงผลที่ดีกว่าได้ ซึ่งระบบกราฟิกส์ในยูนิกซ์นั้นจะต้องพัฒนาบนระบบ X Windows หรือถ้าจะพัฒนาฝั่งไคลเอนต์เป็นบนสภาพแวดล้อมของวินโดว์ 98 ก็ย่อมได้

บรรณานุกรม

[1] W.Richard Stevens, "UNIX Network Programming" , Prentice-Hall International, Inc., 772 p., 1991

[2] Douglass E. Comer & David Stevens, "Internetworking With TCP/IP Volume III", Prentice-Hall International, Inc., 519 p., 1996

[3] John Bloomer, "Power Programming with RPC", O'Reilly & Associates, Inc., 486 p., 1992

[4] Neil Matthew & Richard Stones, "Beginning LINUX Programming", Wrox Press Ltd., 710 p., 1996

[5] Michael Padovano, "Networking Applications on UNIX System V Release 4", Prentice-Hall,Inc., 544 p., 1993

[6] ธันวา ศรีปราโมง, "การเขียนโปรแกรมภาษาซี สำหรับวิศวกรรม" , มหาวิทยาลัยเทคโนโลยีมหานคร , 739 หน้า , 2539

[7] สันติ ศรีลาศักดิ์ และ วรวุฒิ เที่ยงธรรม, "เจาะประเด็นงานเขียนโปรแกรมบนลีนุกซ์" , บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน), 418 หน้า , 2542

[8] www.stardust.com

[9] www.oncrpc.com

ภาคผนวก ก.

โครงสร้างของโปรแกรม

โครงสร้างของโปรแกรมจำลองเครื่อง ATM จะประกอบด้วย 3 ส่วน คือ ส่วนของโปรแกรมฝั่งเซิร์ฟเวอร์ ส่วนของโปรแกรมฝั่งไคลเอนต์ และส่วนของโปรแกรมที่ใช้งานร่วมกัน

1. โปรแกรมฝั่งเซิร์ฟเวอร์

เป็นโปรแกรมที่ต้องทำงานที่ฝั่งเซิร์ฟเวอร์ประกอบด้วยไฟล์หลายไฟล์ที่จะต้องนำมา link เข้าด้วยกัน ซึ่งจะแบ่งไฟล์ทั้งหมดออกเป็น 2 กลุ่มคือ กลุ่มแรกเป็นไฟล์ที่ได้จาก rpcgen compiler ประกอบไปด้วยไฟล์

atm_xdr.c เป็นไฟล์ xdr สำหรับแสดงรูปแบบของข้อมูลที่ใช้ติดต่อสื่อให้เข้าใจตรงกันระหว่างไคลเอนต์และเซิร์ฟเวอร์

atm_svc.c เป็นไฟล์ Stub เซิร์ฟเวอร์

atm.h เป็น header ไฟล์สำหรับ include เข้าไปในโปรแกรมเซิร์ฟเวอร์

กลุ่มสองเป็นไฟล์ที่ได้จากการพัฒนาขึ้นมาประกอบด้วยไฟล์ atmsv.c

ไฟล์ atmsv.c

เป็นไฟล์หลักที่อยู่ฝั่งเซิร์ฟเวอร์ พัฒนาขึ้นสำหรับให้บริการด้าน ตรวจรหัสผ่านและหมายเลขบัญชี การถอนเงิน การโอนเงิน การดูยอดคงเหลือ ในการเขียนโปรแกรมนั้นจะใช้หลักการเขียนตามรูปแบบของการพัฒนาโปรแกรมด้วยกลไก RPC ในไฟล์ atmsv.c มีฟังก์ชันทั้งหมด 5 ฟังก์ชันดังนี้

int check_dest(char *number)

อินพุท: ตัวแปร pointer ชนิด char

เอาท์พุท: ตัวแปรชนิด integer

ทำหน้าที่ เช็ค ID ปลายทางว่ามีอยู่หรือไม่กรณีที่เรียกใช้โพรซีเดอร์โอนเงิน

transfer_resulttype *transfer_1(transfer_moneytype *num_transfer)

อินพุท : ตัวแปรโครงสร้างที่เป็น pointer ประกอบด้วย ID ต้นทาง, ID ปลายทาง และจำนวนเงินที่โอน

เอาท์พุท : ตัวแปรโครงสร้างที่เป็น pointer ประกอบด้วย ID ต้นทาง, ID ปลายทาง, จำนวนเงินที่โอนได้, ยอดคงเหลือในบัญชี และ วันเวลาที่โอน

ทำหน้าที่ : โอนเงินจากบัญชีต้นทางไปยังบัญชีปลายทางที่ระบุ

balance_resulttype *withdraw_1(withdraw_moneytype *num_money)

อินพุท : ตัวแปรโครงสร้างที่เป็น pointer ประกอบด้วย ID ที่ต้องการถอน และจำนวนเงินที่ถอน

เอาท์พุท : ตัวแปรโครงสร้างที่เป็น pointer ประกอบด้วย ID ที่ถอน, ยอดคงเหลือในบัญชี และ วันเวลา ที่ถอนเงิน

ทำหน้าที่ : ถอนเงินจากบัญชีที่ระบุ

balance_resulttype *check_balance_1(char **ID_number)

อินพุท : ตัวแปร string ที่เป็น ID ที่ต้องการทราบยอดคงเหลือ

เอาท์พุท : ตัวแปรโครงสร้างที่เป็น pointer ประกอบด้วย ID , ยอดคงเหลือในบัญชี และ เวลาที่ทำ

ทำหน้าที่ : เช็คยอดจากบัญชีที่ระบุ

usrinfo *validate_1(usrinfo *info_user)

อินพุท : ตัวแปรโครงสร้าง ที่เป็น pointer ประกอบด้วย ID และรหัสผ่าน

เอาท์พุท : ตัวแปรโครงสร้างที่เป็น pointer ประกอบด้วย ID , และรหัสผ่านเหมือนกัน

ทำหน้าที่ : ตรวจสอบ ID และรหัสผ่าน

2. โปรแกรมฝั่งไคลเอนต์

เป็นโปรแกรมที่ต้องทำงานที่ฝั่งไคลเอนต์ระกอบด้วยไฟล์หลายไฟล์ที่จะต้องนำมา link เข้าด้วยกัน ซึ่งจะแบ่งไฟล์ทั้งหมดออกเป็น 2 กลุ่มคือ กลุ่มแรกเป็นไฟล์ที่ได้จาก rpcgen compiler ประกอบไปด้วยไฟล์

atm_xdr.c เป็นไฟล์ xdr สำหรับแสดงรูปแบบของข้อมูลที่ใช้ติดต่อสื่อให้เข้าใจตรงกันระหว่างไคลเอนต์และเซิร์ฟเวอร์

atm_clnt.c เป็นไฟล์ Stub ไคลเอนต์

atm.h เป็น header ไฟล์สำหรับ include เข้าไปในโปรแกรมไคลเอนต์

กลุ่มสองเป็นไฟล์ที่ได้จากการพัฒนาขึ้นมาได้แก่ไฟล์ atmcl.c ซึ่งมีเพียงฟังก์ชัน Main เท่านั้น การทำงานของโปรแกรมนั้นประกอบด้วยส่วนติดต่อกับผู้ใช้และส่วนที่ให้บริการ

3. โปรแกรมที่ใช้งานร่วมกัน

จะแบ่งเป็น 2 ส่วนคือส่วนที่เป็น protocol ที่ใช้ร่วมกันทั้งไคลเอนต์และเซิร์ฟเวอร์ได้แก่ไฟล์ atm.x ไฟล์นี้จะเป็นอินพุทไฟล์ให้แก่ rpcgen และส่วนที่ใช้สำหรับการทำ rollback จะใช้วิธีการติดต่อแบบ socket โดยที่ฝั่งไคลเอนต์จะเป็นเพียงโพรซีเดอร์หนึ่งในฟังก์ชัน Main และที่ฝั่งเซิร์ฟเวอร์จะเป็นไฟล์ rollback.c การทำ rollback นี้จะตอบสนองต่อการ interrupt signal ด้วยการกด ctrl+c

ภาคผนวก ข.

ซอร์สโค้ดโปรแกรมในโครงงานนี้

ซอร์สโค้ดโปรแกรมเซิร์ฟเวอร์ atmsv.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <fcntl.h>
#include <rpc/rpc.h>
#include "atm.h"

FILE    *fp,*fp1,*ftemp,*ftemp_des;
char    lock_file[] = "/home/dech/complete/LCK.test";
int     fildes;
extern  int  errno;

/*function check destination ID*/
int check_dest(number)
char *number;
{
  FILE *file;
  char x4[15],x5[5];
  int  re_check;
  re_check=0;
  file = fopen("customer.dat","r+");

  while (!feof(file))
  {
    fread(&x4,sizeof(x4),1,file);
    fseek(file,0L,SEEK_CUR);
    fread(&x5,sizeof(x5),1,file);
    fputs(x4,stdout);
      if ((strcmp(number,x4))==0)
              re_check=1;
      if(re_check==1) return(1);
      else return(0);
}


transfer_resulttype *
transfer_1(num_transfer)
transfer_moneytype *num_transfer;
{
        static transfer_resulttype result_transfer=NULL;
        static balance dbbl_src=NULL,dbbl_des=NULL;
        char x1[15],x2[15],x3[15],x4[15],x5[15],x6[15];
        long n,n1,n2,n3;
        int i,j,check;
        time_t lowlevel_time;
        n=0;
        (void)time(&lowlevel_time);

        do{
        /*create lock file*/
        fildes = open(lock_file, O_RDWR | O_CREAT | O_EXCL,0444);
        if (fildes == -1)
                printf(" %d - Lock already present\n",getpid());
        else
```

```
{
printf(" %d - I have exclusive access\n",getpid());

/*result_transfer.errno=0;*/
check=check_dest(num_transfer->Destination_id);
fprintf(stderr,"\n%d",check);

if(check==1)
{
 fp=fopen(DATABL,"r+");
 fp1=fopen(DATABL,"r+");
 while(!feof(fp))
 {
        /*read value source ID from database*/
        fread(&x1,sizeof(x1),1,fp);
        dbbl_src.ID_account=malloc(sizeof(x1));
        memcpy(dbbl_src.ID_account,x1,sizeof(x1));
        fseek(fp,50L,SEEK_CUR);  /*move pointer to field balance*/

        /*read value source balance from database*/
        fread(&x2,sizeof(x2),1,fp);
        dbbl_src.Balance=malloc(sizeof(x2));
        memcpy(dbbl_src.Balance,x2,sizeof(x2));

        if((strcmp(num_transfer->ID_account,dbbl_src.ID_account))==0)
        {
                /*move old source balance value to temp.dat*/
                ftemp=fopen(TEMP,"w+");
                        fwrite(&x1,sizeof(x1),1,ftemp);
                        fwrite(&x2,sizeof(x2),1,ftemp);
                fclose(ftemp);

                /*change string to long int*/
                n=atol(dbbl_src.Balance);
                n1=atol(num_transfer->Num_trans);

                /*check source money in database*/
        if (n1 > n)
        {
                fclose(fp);
                (void)close(fildes);
                (void)unlink(lock_file);
                result_transfer.errno=2;
                return(&result_transfer);
        }

                n=n-n1;
        /*debit in database*/
        fseek(fp,-15L,SEEK_CUR);        /*move pointer to field balance*/
        dbbl_src.Balance=ltoa(n);       /*change long int to string*/
        memcpy(x3,dbbl_src.Balance,sizeof(x3));
        fwrite(&x3,sizeof(x3),1,fp);    /*write new value to database*/

/*result of source ID account*/
result_transfer.transfer_resulttype_u.transfer.Balance=malloc(sizeof(x2));
result_transfer.transfer_resulttype_u.transfer.Date_Time=ctime(&lowlevel_time);
result_transfer.transfer_resulttype_u.transfer.Date_Time[30]='\0';
memcpy(result_transfer.transfer_resulttype_u.transfer.Balance,dbbl_src.
Balance,sizeof(x2));

                while(!feof(fp1))
                {
                        /*read value destination ID from database*/
```

```
                                fread(&x4,sizeof(x4),1,fp1);
                                dbbl_des.ID_account=malloc(sizeof(x4));
                                memcpy(dbbl_des.ID_account,x4,sizeof(x4));
                                /*move pointer to field balance*/
                                fseek(fp1,50L,SEEK_CUR);

                                /*read value destination balance from database*/
                                fread(&x5,sizeof(x5),1,fp1);
                                dbbl_des.Balance=malloc(sizeof(x5));
                                memcpy(dbbl_des.Balance,x5,sizeof(x5));

                        if ((strcmp(num_transfer-Destination_id,dbbl_des.ID_account))==0)
                                {
                                /*move old destination balance value to temp_des.dat*/
                                ftemp_des=fopen(TEMP_DS,"w+");
                                fwrite(&x4,sizeof(x4),1,ftemp_des);
                                fwrite(&x5,sizeof(x5),1,ftemp_des);
                                fclose(ftemp_des);

                                        /*change string to long int*/
                                        n2=atol(dbbl_des.Balance);
                                        n3=atol(num_transfer->Num_trans);

                                        n2=n2+n3;       /*add money to destination ID*/
                                        fseek(fp1,-15L,SEEK_CUR);
                                        dbbl_des.Balance=ltoa(n2);
                                        memcpy(x6,dbbl_des.Balance,sizeof(x6));
                                        /*write new value to database*/
                                        fwrite(x6,sizeof(x6),1,fp1);

                                        fclose(fp);
                                        fclose(fp1);
                                        (void)close(fildes);
                                        (void)unlink(lock_file);
                                        result_transfer.errno=0;
                                        return(&result_transfer);
                                } /*if compare destination ID*/
                        }       /*loop while (!feof(fp1))*/
                } /*if compare source ID*/
        } /*loop while (!feof(fp))*/
        } else result_transfer.errno=1;/*check destination ID*/

        fclose(fp);
        fclose(fp1);
        (void)close(fildes);
        (void)unlink(lock_file);
        return(&result_transfer);

        }
        /*delay lock file*/
        for (i=0;i<500;i++)
                for(j=0;j<30000;j++);
        }while(fildes!=0);

        fclose(fp);
        fclose(fp1);
        result_transfer.errno=0;
        return(&result_transfer);
}

balance_resulttype *
withdraw_1(num_money)
withdraw_moneytype *num_money;
```

```
{
        static balance_resulttype result_withdraw=NULL;
        static balance dbbl;
        char x1[15],x2[15],x3[15];
        int x,i,j,found;
        long n,n1;
        char ch;
        time_t lowlevel_time;

        do{
        /*create lock file*/
        fildes = open(lock_file, O_RDWR | O_CREAT | O_EXCL,0444);

                if (fildes == -1)
                        printf(" %d - Lock already present\n",getpid());
                else
                {
                printf(" %d - I have exclusive access\n",getpid());
                n=0;
                found=0;
                x=1;

                /*result_withdraw.errno=0;*/
                fp=fopen(DATABL,"r+");
        while(!feof(fp))
        {
                /*read value ID from database*/
                fread(&x1,sizeof(x1),1,fp);
                dbbl.ID_account=malloc(sizeof(x1));
                memcpy(dbbl.ID_account,x1,sizeof(x1));
                fseek(fp,50L,SEEK_CUR);/*move pointer to field balance*/

                /*read value balance from database*/
                fread(&x2,sizeof(x2),1,fp);
                dbbl.Balance=malloc(sizeof(x2));
                memcpy(dbbl.Balance,x2,sizeof(x2));

                if ((strcmp(num_money->ID_account,dbbl.ID_account))==0)
                {
                        /*move old balance value to temp.dat*/
                        ftemp=fopen(TEMP,"w+");
                        fwrite(&x1,sizeof(x1),1,ftemp);
                        fwrite(&x2,sizeof(x2),1,ftemp);
                        fclose(ftemp);

                        /*change string to long int*/
                        n=atol(dbbl.Balance);
                        n1=atol(num_money->Amount_money);

                        /*check money in database*/
                                if ( n1 > n)
                                {
                                        fclose(fp);
                                        (void)close(fildes);
                                        (void)unlink(lock_file);
                                /*2 is show money not enough*/
                                result_withdraw.errno=2;
                                return(&result_withdraw);
                                }
                                n=n-n1;
                                /*debit in database*/
                                /*move pointer to field balance*/
```

```
                                fseek(fp,-15L,SEEK_CUR);
                                /*change long int to string*/
                                dbbl.Balance=ltoa(n);
                                memcpy(x3,dbbl.Balance,sizeof(x3));
                        fwrite(&x3,sizeof(x3),1,fp);  /*write new value to database*/

        result_withdraw.balance_resulttype_u.balances.Balance=malloc(sizeof(x3));
        memcpy(result_withdraw.balance_resulttype_u.balances.Balance,dbbl.Balance,size
of(x3));
        result_withdraw.balance_resulttype_u.balances.ID_account=malloc(sizeof(x1));
        memcpy(result_withdraw.balance_resulttype_u.balances.ID_account,dbbl.ID_acco
unt,sizeof(x1));
       result_withdraw.balance_resulttype_u.balances.Date_Time=ctime(&lowlevel_time);

                        /*display time*/
                        (char *)time(&lowlevel_time);
                        printf("The date is: %s",ctime(&lowlevel_time));

                        fclose(fp);
                        (void)close(fildes);
                        (void)unlink(lock_file);
                        result_withdraw.errno=0;        /*0 is result complete*/
                        return(&result_withdraw);
                }   /*if compare ID*/
         } /*loop while(!feof(fp))*/

        }  /*else of lock file*/
                for (i=0;i<500;i++)
                for(j=0;j<30000;j++);
        }while(fildes!=0); /*loop do while*/
        fclose(fp);
        return(&result_withdraw);
}

balance_resulttype *
check_balance_1(ID_number)
char **ID_number;
{
        static balance_resulttype result_check_balance=NULL;
        static balance dbbl;
        time_t lowlevel_time;
        char x1[15],x2[15];

        (char *)time(&lowlevel_time);
        printf("The date is: %s",ctime(&lowlevel_time));

        fp=fopen(DATABL,"rb");
        while (!feof(fp))
        {
                /*read value ID from database*/
                fread(&x1,sizeof(x1),1,fp);
                dbbl.ID_account=malloc(sizeof(x1));
                memcpy(dbbl.ID_account,x1,sizeof(x1));
                fseek(fp,50L,SEEK_CUR);/*move pointer to field balance*/

                /*read value balance from database*/
                fread(&x2,sizeof(x2),1,fp);
                dbbl.Balance=malloc(sizeof(x2));
                memcpy(dbbl.Balance,x2,sizeof(x2));

                /*compare ID*/
                if((strcmp(*ID_number,dbbl.ID_account))==0)
                {
```

```
                result_check_balance.balance_resulttype_u.balances.ID_account=malloc
(sizeof(x1));

        memcpy(result_check_balance.balance_resulttype_u.balances.ID_account,dbbl.ID
_account,sizeof(x1));
        result_check_balance.balance_resulttype_u.balances.Balance=malloc(sizeof(x2));
        memcpy(result_check_balance.balance_resulttype_u.balances.Balance,dbbl.Balanc
e,sizeof(x2));
        result_check_balance.balance_resulttype_u.balances.Date_Time=ctime
(&lowlevel_time);
                }
        }
        fclose(fp);
        return(&result_check_balance);
}

usrinfo *
validate_1(info_user)
usrinfo *info_user;
{
        static usrinfo result_validate,dbusr;
        char x1[15],x2[5];

        /*allocate memory*/
        result_validate.ID_account=malloc(sizeof(x1));
        result_validate.Password=malloc(sizeof(x2));
        dbusr.ID_account=malloc(sizeof(x1));
        dbusr.Password=malloc(sizeof(x2));

        fp=fopen(DATAUSR,"rb");
        while (!feof(fp))
        {
                /*read value ID from database*/
                fread(&x1,sizeof(x1),1,fp);
                memcpy(dbusr.ID_account,x1,sizeof(x1));

                /*read value password from database*/
                fread(&x2,sizeof(x2),1,fp);
                memcpy(dbusr.Password,x2,sizeof(x2));

                /*compare ID*/
                if ((strcmp(info_user->ID_account,dbusr.ID_account))==0)
                {
                memcpy(result_validate.ID_account,dbusr.ID_account,sizeof(x1));

                  /*compare password*/
                  if ((strcmp(info_user->Password,dbusr.Password))==0)
                        memcpy(result_validate.Password,dbusr.Password,sizeof(x2));
                        fclose(fp);
                        return(&result_validate);
                }
        }
        fclose(fp);
        return(&result_validate);
}
```

ซอร์สโค้ดโปรแกรมไคลเอนต์ atmcl.c

```
#include <stdio.h>
#include <stdlib.h>
#include <curses.h>
#include <malloc.h>
#include <signal.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <rpc/rpc.h>
#include "atm.h"

#define  SIZE 8192
#define  PORT 0x1233  /*define port communication*/

static void sig_pipe();
void roll_back();
int Flag_check=0;
CLIENT *cl;       /*define client handle*/

WINDOW *win;
char id[15],passwd[5],iwm[15],itm[15],di[15];
usrinfo   info,*result_validate=NULL;
transfer_moneytype     transfer_money;
withdraw_moneytype   withdraw_money;
transfer_resulttype      *result_transfer=NULL;
balance_resulttype      *result_check_balance=NULL,*result_withdraw=NULL;
struct servent *sp;

int  *flag_dest_id=0;
char *host = "161.246.5.240";
extern  int errno;

/*function validate ID and password*/
int validate()
{
        int flag,flag_id,flag_pwd;
        flag=0;flag_id=0;flag_pwd=0;
        if ((strcmp(info.ID_account,result_validate->ID_account))!=0) flag_id=-1;
        if ((strcmp(info.Password,result_validate->Password))!=0) flag_pwd=-1;

                if (flag_id==-1)
                   {
                   mvwprintw(win,13,10,"ID_account wrong");
                   mvwprintw(win,13,40,"%s",result_validate->ID_account);
                   mvwprintw(win,16,10,"Press return key for reagain");
                   flag=flag_id;
                   wgetch(win);
                   return(flag);
                   }
                   else if (flag_pwd==-1)
                       {
                         mvwprintw(win,13,10,"Password wrong");
                       mvwprintw(win,16,10,"Press return key for reagain");
                         flag=flag_pwd;
                         wgetch(win);
                         return(flag);
                         }
}
```

```
main(argc,argv)
int argc;
char *argv[];
{
        int  i,count=1;
        struct timeval tv;
        char  *server;
        int  ck=0;
        char ch;

                if (argc!=2)
                {
                        fprintf(stderr,"usage: %s ip of hostname\n",argv[0]);
                        exit(1);
                }

        /*create client handle*/
        server=argv[1];
        cl=clnt_create(server,ATMPROG,ATMVERS,"tcp");

                if (cl == NULL)
                {
                        clnt_pcreateerror(server);
                        exit(1);
                }

        initscr();

        /*loop do while input password */
        do{
        win = newwin(20,70,2,4);
        box(win,'|','-');
        mvwprintw(win,3,20,"Welcome to ATM Simulation");
        mvwprintw(win,17,50,"Ctrl C : Cancel");
        mvwprintw(win,6,10,"Please input ID Account: ");
        noecho();

                wmove(win,6,35);
                i=0;
                while((id[i] = wgetch(win))!='\n')
                {
                  wprintw(win,"%c",id[i]);
                  i++;
                }
                id[i]='\0';

                mvwprintw(win,8,10,"Please input Password: ");
                wmove(win,8,35);
                i=0;
                while((passwd[i] = wgetch(win))!='\n')
                {
                  wprintw(win,"*");
                  i++;
                }
                passwd[i]='\0';

        /*input value to procedure on remote*/
        info.ID_account=id;
        info.Password=passwd;

        /*call procedure validate on remote*/
        result_validate=validate_1(&info,cl);
        ck=validate();
```

```
        echo();
        count++;
        } while ((ck==-1) && (count<4));


    if ((count<5) && (ck!=-1))
    {
        wclear(win);
        box(win,'|','-');
        mvwprintw(win,3,20,"%s","Welcome to ATM Simulation");
        mvwprintw(win,6,10,"%s","Select Service");
        mvwprintw(win,8,10,"%s","1.Check Balance");
        mvwprintw(win,10,10,"%s","2.Withdrawal");
        mvwprintw(win,12,10,"%s","3.Transfer");
        mvwprintw(win,17,50,"Ctrl C : Cancel");
        mvwprintw(win,14,10,"%s","Please input service: ");
        wmove(win,14,32);

        /*select services*/
        ch=wgetch(win);wprintw(win,"%c",ch);
        switch(ch) {

                /*check balance*/
                case '1': wclear(win);
                        box(win,'|','-');
                        mvwprintw(win,3,20,"%s","Welcome to ATM Simulation");
                        mvwprintw(win,6,10,"%s","Check Balance Service");
                        mvwprintw(win,17,50,"Ctrl C : Cancel");

                        /*call procedure check balance on remote*/
                        result_check_balance=check_balance_1(&info.ID_account,cl);
                        /*show result from remote*/
                        mvwprintw(win,8,10,"Balance = %s",result_check_balance->
balance_resulttype_u.balances.Balance);
                        mvwprintw(win,10,10,"ID Account = %s",result_check_balance->
balance_resulttype_u.balances.ID_account);
                        mvwprintw(win,12,10,"Date/Time %s",result_check_balance->
balance_resulttype_u.balances.Date_Time);
                        box(win,'|','-');
                        break;

                /*withdraw*/
                case '2': wclear(win);
                        box(win,'|','-');
                        mvwprintw(win,3,20,"%s","Welcome to ATM Simulation");
                        mvwprintw(win,6,10,"%s","Withdrawal Service");
                        mvwprintw(win,17,50,"Ctrl C : Cancel");
                        mvwprintw(win,8,10,"%s","Please input withdraw money: ");
                        wgetstr(win,iwm);

                        /*input value to procedure on remote*/
                        withdraw_money.ID_account=id;
                        withdraw_money.Amount_money=iwm;

                        /*define and set wait time*/
                        tv.tv_sec=25;
                        tv.tv_usec=0;
                        clnt_control(cl,CLSET_TIMEOUT,&tv);

                        /*check signal Ctrl-C*/
                        signal(SIGINT,sig_pipe);
                        count=0;
```

```
                        do{
                                result_withdraw=withdraw_1(&withdraw_money,cl);
                                if (result_withdraw==NULL)
                                        count++;
                        }while ((count<2) && (result_withdraw==NULL));

                        /*check cancel*/
                        if(Flag_check==0)
                        {
                        /*0 is result return complete*/
                        /*2 is money not enough*/
                        if (result_withdraw->errno==0){
                                mvwprintw(win,12,10,"Balance = %s",result_withdraw->
balance_resulttype_u.balances.Balance);
                                mvwprintw(win,14,10,"ID Account = %s",result_withdraw->
balance_resulttype_u.balances.ID_account);
                                mvwprintw(win,16,10,"Date/Time %s",result_withdraw->
balance_resulttype_u.balances.Date_Time);
                                box(win,'|','-');
                                }
                                else mvwprintw(win,15,10,"Money not enough");
                        }
                        else mvwprintw(win,15,10,"Cancel");
                        break;

                /*transfer*/
                case '3':
                        wclear(win);
                        box(win,'|','-');
                        mvwprintw(win,3,20,"%s","Welcome to ATM Simulation");
                        mvwprintw(win,6,10,"%s","Transfer Service");
                        mvwprintw(win,17,50,"Ctrl C : Cancel");
                        mvwprintw(win,8,10,"%s","Please input Destination ID
                        Account:");
                        wgetstr(win,di);
                        mvwprintw(win,10,10,"%s","Please input Transfer money: ");
                        wgetstr(win,itm);

                        /*input value to procedure on remote*/
                        transfer_money.ID_account=id;
                        transfer_money.Destination_id=di;
                        transfer_money.Num_trans=itm;
                        count=0;

                        /*define and set wait time*/
                        tv.tv_sec=25;
                        tv.tv_usec=0;
                        clnt_control(cl,CLSET_TIMEOUT,&tv);

                        /*check signal Ctrl-C*/
                        signal(SIGINT,sig_pipe);
                        do{
                                result_transfer=transfer_1(&transfer_money,cl);
                                if (result_transfer==NULL)
                                        count++;
                        }while ((count<2) && (result_transfer==NULL));

                       /*check cancel*/
                       if(Flag_check==0)
                       {
                        /*0 is result return complete*/
                        /*1 is destination ID wrong*/
                        /*2 is money not enough*/
```

```
                        if (result_transfer->errno==0)
                        {
                                mvwprintw(win,12,10,"Balance = %s",result_transfer->
transfer_resulttype_u.transfer.Balance);
                                mvwprintw(win,13,10,"From ID account: %s",id);
                                mvwprintw(win,14,10,"To ID account:   %s",di);
                                mvwprintw(win,15,10,"Date/Time %s",result_transfer->
transfer_resulttype_u.transfer.Date_Time);
                                box(win,'|','-');
                           }
                        else if (result_transfer->errno==1)
                                mvwprintw(win,15,10,"Destination ID account wrong");
                        else  if (result_transfer->errno==2)
                                mvwprintw(win,15,10,"Money not enough");
                        }
                        else mvwprintw(win,15,10,"Cancel");
                        break;
                       }
                wrefresh(win);
                endwin();
        }
        wrefresh(win);
        endwin();
}

/*function rollback*/
void roll_back()
{
        int sd,i,k;
        char *ch2,*ch1,ch4[8];
        struct sockaddr_in client_addr;
        struct hostent *hp;
        char buf[SIZE];
        int nread;
        ch1 = transfer_money.ID_account;
        ch2 = transfer_money.Destination_id;
        bzero(ch4,sizeof(ch4));
        for(k=0;k<4;k++)
        ch4[k] = ch1[k];
        for(i=0;i<4;i++)
        ch4[i+k] = ch2[i];

        /* clear memory of structure */
        bzero((char *)&client_addr,sizeof(client_addr));

        /* specific for communication */
        client_addr.sin_family = AF_INET;
        client_addr.sin_addr.s_addr = inet_addr("161.246.5.240");
        client_addr.sin_port = htons(PORT);

        /* create socket */
           if((sd=socket(AF_INET,SOCK_STREAM,0)) < 0)
                   {
                   perror("create fail\n");
                   exit(1);
                  }

     /* connection request to host */
          if((connect(sd,(struct client_addr *)&client_addr,sizeof(client_addr))) < 0)
             {
                   perror("connect to host fail\n");
                   exit(1);
              }
```

```
        if(send(sd,ch4,sizeof(ch4),0)==-1){
      perror("send");
        exit(1);
            }
         if(recv(sd,buf,SIZE,0)==-1){
            perror("recv");
            exit(1);
            }
                mvwprintw(win,15,10,"%s",buf);
              close(sd);
}

/*function check signal interrupt*/
static void sig_pipe(t)
int t;
{
   Flag_check = 1;
   /*call function roll_back on local system*/
  /*roll_back();*/
   clnt_destroy(cl);
   roll_back();
   return;
}
```

ซอร์สโค้ดโปรแกรมโรลแบ็ค rollback.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <netdb.h>

#define PORT 0x1233
#define SIZE 8192
char temp_file[]="temp.dat";

main(argc, argv)
int argc;
char *argv[];
 {
  FILE *fp,*fp1,*ftemp,*ftemp_des;
  struct sockaddr_in sin;
  struct sockaddr_in pin;
  char buf[SIZE],x0[15],x1[15],x2[15],x3[15],x4[15],x5[15],x6[15];
  char *ch1,*ch2;
  int i,count,s,ns;
  ch1="1";ch2="1";
  count=0;
        if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("tomd: socket");
        exit(1);
        }
   bzero(&sin, sizeof(sin));
   sin.sin_family=AF_INET;
   sin.sin_port=htons(PORT);
   sin.sin_addr.s_addr=htonl(INADDR_ANY);

   if (bind(s, (struct sockaddr *) &sin, sizeof(sin)) == -1) {
        perror("tomd: bind");
        exit(1);
        }

   if (listen(s, 5) == -1) {
        perror("tomd: listen");
        exit(1);
        }

while(1){
  if ((ns = accept(s, 0, 0)) == -1) {
     perror("tomd: accept");
     exit(1);
       }
  if(recv(ns,buf,sizeof(buf),0)==-1){
     perror("recv");
     exit(1);
    }
  for(i=0;i<strlen(buf);i++)
   {
    if(i<4) ch1[i] = buf[i];
    else ch2[i-4] = buf[i];
   }
   fputs(buf,stdout);
   ftemp=fopen("temp.dat","r+");
           fread(&x1,sizeof(x1),1,ftemp);
           fread(&x2,sizeof(x2),1,ftemp);
     fclose(ftemp);
```

```
        fputs(x1,stdout);
        if ((strcmp(ch1,x1))==0) fputs("string ok",stdout);

        ftemp_des=fopen("temp_des.dat","r+");
              fread(&x5,sizeof(x5),1,ftemp_des);
              fread(&x6,sizeof(x6),1,ftemp_des);
        fclose(ftemp_des);

        fp=fopen("balance.dat","r+");
        while(!feof(fp) && (count<2))
        {
              fread(&x3,sizeof(x3),1,fp);
              fseek(fp,50L,SEEK_CUR);
              fread(&x4,sizeof(x4),1,fp);

              if ((strcmp(x1,x3))==0)
              {
                    fputs("ok",stdout);
                    fseek(fp,-15L,SEEK_CUR);
                    fwrite(&x2,sizeof(x2),1,fp);
                    count++;

              }
              else if ((strcmp(x5,x3))==0)
                    {
                    fputs("ok2",stdout);
                    fseek(fp,-15L,SEEK_CUR);
                    fwrite(&x6,sizeof(x6),1,fp);
                    count++;
                    }

        }
        fclose(fp);
    if(send(ns,"ok",strlen("ok"),0)==-1){
      perror("send");
      exit(1);
     }
    close(ns);
    count=0;
  }
 close(s);
}
```

ซอร์สโค้ดโปรแกรม atm.x และไฟล์ต่างๆที่เกิดจาก rpcgen คอมไพลเลอร์

```
%#define DATAUSR "customer.dat"
%#define DATABL "balance.dat"
%#define TEMP "temp.dat"
%#define TEMP_DS "temp_des.dat"

struct usrinfo{
        string ID_account<15>;
        string Password<5>;
        };

struct balance{
        string ID_account<15>;
        string Firstname<20>;
        string Lastname<30>;
        string Balance<15>;
        };

struct balance_type{
        string  Balance<15>;
        string  ID_account<15>;
        string  Date_Time<>;
        int     Flag;
        };

struct withdraw_moneytype{
        string ID_account<15>;
        string Amount_money<15>;
        };

struct transfer_moneytype{
        string ID_account<15>;
        string Destination_id<15>;
        string Num_trans<15>;
        };

struct transfer_type{
        string ID_account<15>;
        string Destination_id<15>;
        string Num_trans<15>;
        string Balance<15>;
        string Firstname_des<20>;
        string Lastname_des<30>;
        string Date_Time<>;
        int    Flag_dest_id;
        int    Flag;

        };

union transfer_resulttype switch(int errno){
    case 0:
        transfer_type transfer;
      default:
        void;
        };

union balance_resulttype switch(int errno){
    case 0:
        balance_type balances;
      default:
        void;
```

```
        };

program ATMPROG {
        version ATMVERS {
                usrinfo VALIDATE(usrinfo)=1;
                balance_resulttype CHECK_BALANCE(string) = 2;
                balance_resulttype WITHDRAW(withdraw_moneytype) = 3;
                transfer_resulttype TRANSFER(transfer_moneytype) = 4;
                }=1;
        }=123456788;
```

ซอร์สโค้ดโปรแกรม atm.h

```
#include <rpc/types.h>

#define DATAUSR "customer.dat"
#define DATABL "balance.dat"
#define TEMP "temp.dat"
#define TEMP_DS "temp_des.dat"

struct usrinfo {
        char *ID_account;
        char *Password;
};
typedef struct usrinfo usrinfo;
bool_t xdr_usrinfo();

struct balance {
        char *ID_account;
        char *Firstname;
        char *Lastname;
        char *Balance;
};
typedef struct balance balance;
bool_t xdr_balance();

struct balance_type {
        char *Balance;
        char *ID_account;
        char *Date_Time;
        int Flag;
};
typedef struct balance_type balance_type;
bool_t xdr_balance_type();

struct withdraw_moneytype {
        char *ID_account;
        char *Amount_money;
};
typedef struct withdraw_moneytype withdraw_moneytype;
bool_t xdr_withdraw_moneytype();

struct transfer_moneytype {
        char *ID_account;
        char *Destination_id;
        char *Num_trans;
};
typedef struct transfer_moneytype transfer_moneytype;
bool_t xdr_transfer_moneytype();

struct transfer_type {
```

```
        char *ID_account;
        char *Destination_id;
        char *Num_trans;
        char *Balance;
        char *Firstname_des;
        char *Lastname_des;
        char *Date_Time;
        int Flag_dest_id;
        int Flag;
};
typedef struct transfer_type transfer_type;
bool_t xdr_transfer_type();

struct transfer_resulttype {
        int errno;
        union {
                transfer_type transfer;
        } transfer_resulttype_u;
};
typedef struct transfer_resulttype transfer_resulttype;
bool_t xdr_transfer_resulttype();

struct balance_resulttype {
        int errno;
        union {
                balance_type balances;
        } balance_resulttype_u;
};
typedef struct balance_resulttype balance_resulttype;
bool_t xdr_balance_resulttype();

#define ATMPROG ((u_long)123456788)
#define ATMVERS ((u_long)1)
#define VALIDATE ((u_long)1)
extern usrinfo *validate_1();
#define CHECK_BALANCE ((u_long)2)
extern balance_resulttype *check_balance_1();
#define WITHDRAW ((u_long)3)
extern balance_resulttype *withdraw_1();
#define TRANSFER ((u_long)4)
extern transfer_resulttype *transfer_1();
```

ซอร์สโค้ดโปรแกรม atm_clnt.c

```
#include <rpc/rpc.h>
#include "atm.h"
#include <time.h>

#ifdef hpux

#ifndef NULL
#define NULL  0
#endif  /* NULL */

#endif  /* hpux */
#define DATAUSR "customer.dat"
#define DATABL "balance.dat"
#define TEMP "temp.dat"
#define TEMP_DS "temp_des.dat"

/* Default timeout can be changed using clnt_control() */
```

```
static struct timeval TIMEOUT = { 25, 0 };

usrinfo *
validate_1(argp, clnt)
        usrinfo *argp;
        CLIENT *clnt;
{
        static usrinfo res;

        bzero((char *)&res, sizeof(res));
        if (clnt_call(clnt, VALIDATE, xdr_usrinfo, argp, xdr_usrinfo, &res, TIMEOUT) !=
RPC_SUCCESS) {
                return (NULL);
        }
        return (&res);
}

balance_resulttype *
check_balance_1(argp, clnt)
        char **argp;
        CLIENT *clnt;
{
        static balance_resulttype res;

        bzero((char *)&res, sizeof(res));
        if (clnt_call(clnt, CHECK_BALANCE, xdr_wrapstring, argp, xdr_balance_resulttype,
&res, TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&res);
}

balance_resulttype *
withdraw_1(argp, clnt)
        withdraw_moneytype *argp;
        CLIENT *clnt;
{
        static balance_resulttype res;

        bzero((char *)&res, sizeof(res));
        if (clnt_call(clnt, WITHDRAW, xdr_withdraw_moneytype, argp,
xdr_balance_resulttype, &res, TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&res);
}

transfer_resulttype *
transfer_1(argp, clnt)
        transfer_moneytype *argp;
        CLIENT *clnt;
{
        static transfer_resulttype res;

        bzero((char *)&res, sizeof(res));
        if (clnt_call(clnt, TRANSFER, xdr_transfer_moneytype, argp,
xdr_transfer_resulttype, &res, TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&res);
}
```

ซอร์สโค้ดโปรแกรม atm_svc.c

```
#include <stdio.h>
#include <rpc/rpc.h>
#include "atm.h"
#define DATAUSR "customer.dat"
#define DATABL "balance.dat"
#define TEMP "temp.dat"
#define TEMP_DS "temp_des.dat"

static void atmprog_1();

main()
{
        register SVCXPRT *transp;

        (void) pmap_unset(ATMPROG, ATMVERS);

        transp = svcudp_create(RPC_ANYSOCK);
        if (transp == NULL) {
                fprintf(stderr, "cannot create udp service.");
                exit(1);
        }
        if (!svc_register(transp, ATMPROG, ATMVERS, atmprog_1, IPPROTO_UDP)) {
                fprintf(stderr, "unable to register (ATMPROG, ATMVERS, udp).");
                exit(1);
        }

        transp = svctcp_create(RPC_ANYSOCK, 0, 0);
        if (transp == NULL) {
                fprintf(stderr, "cannot create tcp service.");
                exit(1);
        }
        if (!svc_register(transp, ATMPROG, ATMVERS, atmprog_1, IPPROTO_TCP)) {
                fprintf(stderr, "unable to register (ATMPROG, ATMVERS, tcp).");
                exit(1);
        }

        svc_run();
        fprintf(stderr, "svc_run returned");
        exit(1);
        /* NOTREACHED */
}

static void
atmprog_1(rqstp, transp)
        struct svc_req *rqstp;
        register SVCXPRT *transp;
{
        union {
                usrinfo validate_1_arg;
                char *check_balance_1_arg;
                withdraw_moneytype withdraw_1_arg;
                transfer_moneytype transfer_1_arg;
        } argument;
        char *result;
        bool_t (*xdr_argument)(), (*xdr_result)();
        char *(*local)();

        switch (rqstp->rq_proc) {
        case NULLPROC:
                (void) svc_sendreply(transp, xdr_void, (char *)NULL);
```

```
                return;

        case VALIDATE:
                xdr_argument = xdr_usrinfo;
                xdr_result = xdr_usrinfo;
                local = (char *(*)()) validate_1;
                break;

        case CHECK_BALANCE:
                xdr_argument = xdr_wrapstring;
                xdr_result = xdr_balance_resulttype;
                local = (char *(*)()) check_balance_1;
                break;

        case WITHDRAW:
                xdr_argument = xdr_withdraw_moneytype;
                xdr_result = xdr_balance_resulttype;
                local = (char *(*)()) withdraw_1;
                break;

        case TRANSFER:
                xdr_argument = xdr_transfer_moneytype;
                xdr_result = xdr_transfer_resulttype;
                local = (char *(*)()) transfer_1;
                break;

        default:
                svcerr_noproc(transp);
                return;
        }
        bzero((char *)&argument, sizeof(argument));
        if (!svc_getargs(transp, xdr_argument, &argument)) {
                svcerr_decode(transp);
                return;
        }
        result = (*local)(&argument, rqstp);
        if (result != NULL && !svc_sendreply(transp, xdr_result, result)) {
                svcerr_systemerr(transp);
        }
        if (!svc_freeargs(transp, xdr_argument, &argument)) {
                fprintf(stderr, "unable to free arguments");
                exit(1);
        }
        return;
}
```

ซอร์สโค้ดโปรแกรม atm_xdr.c

```
#include <rpc/rpc.h>
#include "atm.h"
#define DATAUSR "customer.dat"
#define DATABL "balance.dat"
#define TEMP "temp.dat"
#define TEMP_DS "temp_des.dat"

bool_t
xdr_usrinfo(xdrs, objp)
        XDR *xdrs;
        usrinfo *objp;
{
        if (!xdr_string(xdrs, &objp->ID_account, 15)) {
                return (FALSE);
```

```
        }
        if (!xdr_string(xdrs, &objp->Password, 5)) {
                return (FALSE);
        }
        return (TRUE);
}

bool_t
xdr_balance(xdrs, objp)
        XDR *xdrs;
        balance *objp;
{
        if (!xdr_string(xdrs, &objp->ID_account, 15)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Firstname, 20)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Lastname, 30)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Balance, 15)) {
                return (FALSE);
        }
        return (TRUE);
}

bool_t
xdr_balance_type(xdrs, objp)
        XDR *xdrs;
        balance_type *objp;
{
        if (!xdr_string(xdrs, &objp->Balance, 15)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->ID_account, 15)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Date_Time, ~0)) {
                return (FALSE);
        }
        if (!xdr_int(xdrs, &objp->Flag)) {
                return (FALSE);
        }
        return (TRUE);
}

bool_t
xdr_withdraw_moneytype(xdrs, objp)
        XDR *xdrs;
        withdraw_moneytype *objp;
{
        if (!xdr_string(xdrs, &objp->ID_account, 15)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Amount_money, 15)) {
                return (FALSE);
        }
        return (TRUE);
}

bool_t
xdr_transfer_moneytype(xdrs, objp)
```

```
        XDR *xdrs;
        transfer_moneytype *objp;
{
        if (!xdr_string(xdrs, &objp->ID_account, 15)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Destination_id, 15)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Num_trans, 15)) {
                return (FALSE);
        }
        return (TRUE);
}

bool_t
xdr_transfer_type(xdrs, objp)
        XDR *xdrs;
        transfer_type *objp;
{
        if (!xdr_string(xdrs, &objp->ID_account, 15)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Destination_id, 15)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Num_trans, 15)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Balance, 15)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Firstname_des, 20)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Lastname_des, 30)) {
                return (FALSE);
        }
        if (!xdr_string(xdrs, &objp->Date_Time, ~0)) {
                return (FALSE);
        }
        if (!xdr_int(xdrs, &objp->Flag_dest_id)) {
                return (FALSE);
        }
        if (!xdr_int(xdrs, &objp->Flag)) {
                return (FALSE);
        }
        return (TRUE);
}

bool_t
xdr_transfer_resulttype(xdrs, objp)
        XDR *xdrs;
        transfer_resulttype *objp;
{
        if (!xdr_int(xdrs, &objp->errno)) {
                return (FALSE);
        }
        switch (objp->errno) {
        case 0:
                if (!xdr_transfer_type(xdrs, &objp->transfer_resulttype_u.transfer)) {
                        return (FALSE);
                }
```

```
                break;
        }
        return (TRUE);
}

bool_t
xdr_balance_resulttype(xdrs, objp)
        XDR *xdrs;
        balance_resulttype *objp;
{
        if (!xdr_int(xdrs, &objp->errno)) {
                return (FALSE);
        }
        switch (objp->errno) {
        case 0:
                if (!xdr_balance_type(xdrs, &objp->balance_resulttype_u.balances)) {
                        return (FALSE);
                }
                break;
        }
        return (TRUE);
}
```