ระบบแฟ้มข้อมูลแบบกระจายโดยใช้ IPC

DISTRIBUTED FILE SYSTEM WITH IPC

โดย

1. นายทรงเดช เย็นทรัพย์ 36013151
2. นายประพนธ์ อุรุเอกโอพาร 36013159

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา

วิศวกรรมศาสตร์บัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2538

ปริญญานิพนธ์ปีการศึกษา 2538

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง แฟ้มข้อมูลแบบกระจายผ่าน IPC

ผู้จัดทำ

1. นาย ทรงเดช เย็นทรัพย์ 36013151
2. นาย ประพนธ์ อุรุเอกโอฬาร 36013159

.....อาจารย์ที่ปรึกษา

(ดร. บุญธีร์ เครือตราชู)

ระบบแฟ้มข้อมูลแบบกระจายโดยใช้ IPC

นายทรงเดช เย็นทรัพย์
นายประพนธ์ อุรุเอกโอฬาร
อาจารย์ที่ปรึกษา ดร. บุญธีร์ เครือตราชู
ปีการศึกษา 2538

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของโครงงานระบบแฟ้มข้อมูลแบบกระจายโดยใช้ IPC เป็นระบบให้บริการแฟ้มข้อมูลแก่ผู้ใช้บริการ โดยแฟ้มข้อมูลจะกระจายอยู่บนโหนดต่างๆภายในเครือข่ายคอมพิวเตอร์ และจะถูกเรียกใช้งานจากผู้ใช้บริการที่อยู่บนโหนดต่างๆในเครือข่ายคอมพิวเตอร์ โดยผู้ใช้บริการจะไม่ต้องทราบเลยว่าแฟ้มข้อมูลที่ตนเองเรียกใช้อยู่นั้นจัดเก็บไว้ที่ใด ในปริญญานิพนธ์นี้จะได้กล่าวถึงความเป็นมา วัตถุประสงค์ ทฤษฎีที่เกี่ยวข้อง หลักการทำงาน ประโยชน์ที่จะได้รับในการพัฒนาระบบ

DISTRIBUTED FILE SYSTEM WITH IPC

Mr. Songdej Yensap

Mr. Prapon Uruak-olarn

Advisor : Dr. Boontee Kruatrachue

Education year 1995

ABSTRACT

Distributed file system with IPC is the system that serves files to clients. Files are kept on several node in computer network. These files are served to clients on several node in computer network. Files are network transparence. The thesis deals with backgrounds, objectives, related theorem, principles, benefits of system developement.

สารบัญ

สารบัญ (ต่อ)

สารบัญรูปภาพ

บทที่ 1
บทนำ

ปัจจุบันความต้องการในการใช้งานระบบฐานข้อมูลขนาดเล็กได้เพิ่มขึ้นอย่างมากและมีระบบฐานข้อมูลขนาดเล็กเพื่อรองรับความต้องการเหล่านั้นพอสมควร แต่เป็นลักษณะทำงานเดี่ยวทั้งหมด แต่ในระบบฐานข้อมูลแบบกระจายจะมีเฉพาะระบบที่ใหญ่ ซึ่งมีประสิทธิภาพสูงและมีราคาแพงเกินความสามารถของหน่วยงานขนาดเล็ก ที่มีความจำเป็นต้องใช้ระบบฐานข้อมูลแบบกระจาย จึงได้แนวคิดในการพัฒนาระบบฐานข้อมูลแบบกระจายขนาดเล็ก ที่มีความสามารถจำกัด ง่ายในการดูแลรักษาและใช้งานระบบ

วัตถุประสงค์

1. เพื่อศึกษาหลักการทำงานของการส่งผ่านข้อมูลบนเครือข่ายคอมพิวเตอร์
2. เพื่อพัฒนาโปรแกรมจัดการฐานข้อมูลแบบกระจายอย่างง่ายๆ
3. เพื่อใช้เป็นแนวทางในการประยุกต์ใช้งานการเขียนโปรแกรมผ่านเครือข่ายคอมพิวเตอร์

บทที่ 2

ทฤษฎี

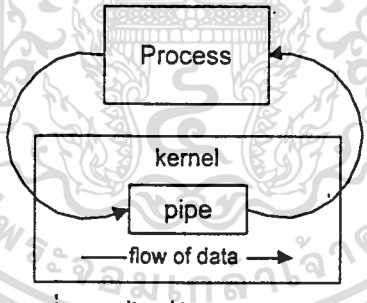
2.1 Interprocess communication

Interprocess communication : IPC คือการติดต่อกันระหว่างโปรเซสภายในเครื่องเดียวกัน ในที่นี้จะกล่าวเฉพาะไปป์ (Pipes) และเมสเซสคิว (Message Queue) เท่านั้น (เนื้อหาเพิ่มเติมใน [1] W. Richard Stevents)

2.1.1 Pipes

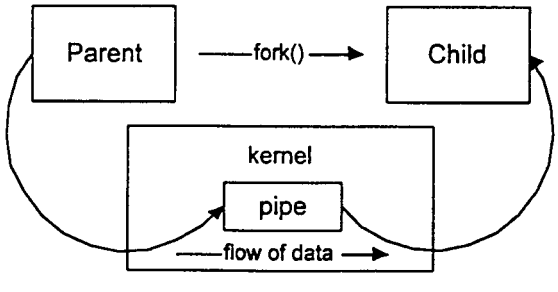
ไปป์ คือลักษณะการติดต่อกันแบบทางเดียวภายในโปรเซสเดียวกัน หรือระหว่างโปรเซสที่เป็นพ่อลูกกัน (parent - child) ที่เกิดจากการ fork() เพราะเมื่อไปป์ถูกสร้างขึ้นมาจะมีหมายเลขประจำไปป์นั้นซึ่งจะรู้หมายเลขนี้เฉพาะโปรเซสที่สร้างขึ้นมาเท่านั้น และเมื่อมีการ fork() โปรเซสลูก ก็จะสามารถรู้หมายเลขประจำไปป์นี้เช่นกัน โดยปกติจะใช้ติดต่อกันระหว่างโปรเซสพ่อและลูก

ไปป์ถูกสร้างโดยเรียก pipe() โดยจะได้หมายเลขประจำไปป์มา 2 ค่า สำหรับอ่านและเขียน ตามลำดับ ดังรูป



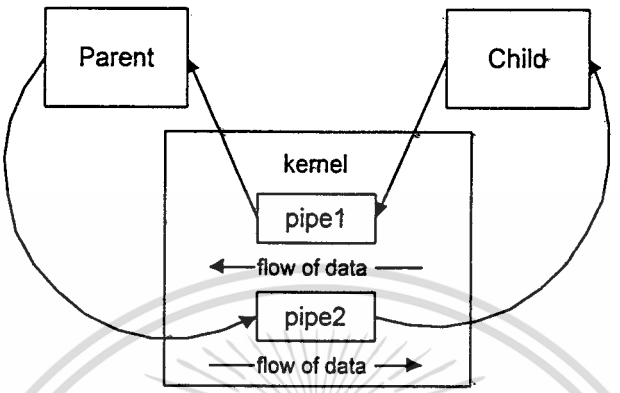
รูปที่ 2.1.1 ไปป์ในโปรเซสเดียว

เมื่อมีการ fork() โปรเซสลูกจะรู้จักหมายเลขไปป์ที่สร้างขึ้น โปรเซสพ่อจะปิดช่องการอ่านโดยเรียก close() โปรเซสลูกจะปิดช่องการเขียน ทำให้โปรเซสพ่อสามารถติดต่อไปยังโปรเซสลูกได้ ดังรูป



รูปที่ 2.1.2 การติดต่อทางเดียวระหว่างโปรเซส

แต่โปรเซสลูกจะติดต่อกลับไม่ได้ ดังนั้นจึงมีการสร้างไปป์ขึ้น 2 ครั้งและทำในลักษณะตรงข้ามกัน จะได้การติดต่อแบบสองทาง ดังรูป



รูปที่ 2.1.3 การติดต่อสองทางระหว่างโปรเซส

เมื่อมีการสร้างไปป์ให้สามารถติดต่อได้ทั้งสองทางแล้วก็สามารถส่งข้อมูลถึงกันได้โดยใช้คำสั่ง read() , write() ได้ตามปกติ

สามารถลำดับการสร้างไปป์สำหรับการติดต่อระหว่างโปรเซสพ่อและลูกได้ดังนี้

1. สร้างไปป์ที่ 1 และ 2 โดยเรียก pipe() 2 ครั้ง
2. สร้างโปรเซสลูก โดยเรียก fork()
3. โปรเซสพ่อปิดการช่องอ่านไปป์ 1 และ ปิดช่องการเขียนไปป์ 2
4. โปรเซสลูกปิดการช่องเขียนไปป์ 1 และ ปิดการช่องอ่านไปป์ 2

คำสั่งเฉพาะที่เกี่ยวกับไปป์มีเพียงคำสั่งเดียว คือ pipe() (นอกนั้นเป็นคำสั่งมาตรฐาน เช่น read(), write(), close() และคำสั่งชั้นสูงแต่จะไม่กล่าวถึง คือ popen(), pclose() สามารถอ่านได้จาก [1] W. Richard Stevents) ซึ่งมีรายละเอียดดังนี้

```
int pipe( int *pipefd[]);
```

โดย *pipefd*[0] เป็นช่องสำหรับอ่าน และ *pipefd*[1] เป็นช่องสำหรับเขียน

2.1.2 Message Queues

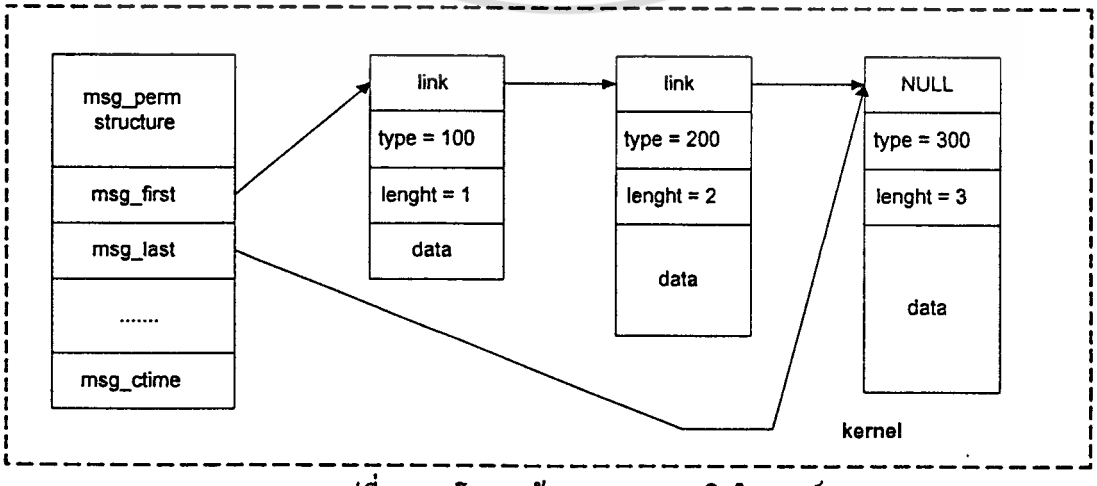
เมสเซสคิวเป็นการติดต่อกันระหว่างโปรเซสต่างๆ มีข้อเด่น คือ สามารถเลือกข้อมูลที่ต้องการได้ โดยกำหนดหมายเลขประจำเมสเซสและจะทำการอ่านเมสเซสทั้งเมสเซสขึ้นมาจากคิว ซึ่งจะทำให้เกิดความสะดวกในการเขียนโปรแกรมยิ่งขึ้นเมื่อเปรียบเทียบกับการใช้ไปป์

โครงสร้างของเมสเซสคิว

```
#include<sys/types.h>
#include<sys/ipc.h>              /* defines the ipc_perm structure */
struct msqid_ds{
  struct ipc_perm msg_perm;  /* operation permission struct */
  struct msg      *msg_first;   /* ptr to first message on q */
  struct msg      *msg_last;    /* ptr to last message on q */
  ushort          msg_cbytes;  /* current # bytes on q */
  ushort          msg_qnum;    /* current # of message on q */
  ushort          msg_qbytes;  /* max # of bytes allowed on q */
  ushort          msg_lspid;    /* pid of last msgsnd */
  ushort          msg_lrpid;    /* pid of last msgrcv */
  time_t          msg_stime    /* time of last msgsnd */
  time_t          msg_rtime    /* time of last msgrcv */
  time_t          msg_ctime    /* time of last msgctl ( that changed the above) */
}
```

ทุกๆเมสเซสคิวจะต้องประกอบไปด้วยแอตทริบิวต์ (Attributes) ต่างๆดังนี้

- long integer type;
- length of data portion of the message (can be zero)
- data (if the length is greater than zero)



รูปที่ 2.1.4 โครงสร้างของเมสเซสคิวในเคอร์เนล

System call ที่สำคัญเกี่ยวกับเมสเซสคิว

int msgget(key_t , int msgflag); เป็นการสร้างเมสเซสหรือเปิดใช้งานเมสเซสที่มีอยู่แล้ว โดยถ้าฝ่ายเซอร์เวอร์ (server) เป็นตัวเรียกฟังก์ชั่นนี้ก็จะเป็นการสร้างเมสเซสขึ้นมาใหม่ แต่ถ้า ไคลแอนต์ (client) เป็นตัวเรียกก็จะเป็นการเปิดใช้งานเมสเซส

int msgsnd(int msqid , struct msgbuf *ptr , int length , int flag); เป็นฟังก์ชั่นที่เรียกใช้เมื่อต้องการส่งเมสเซสลงไปในคิว

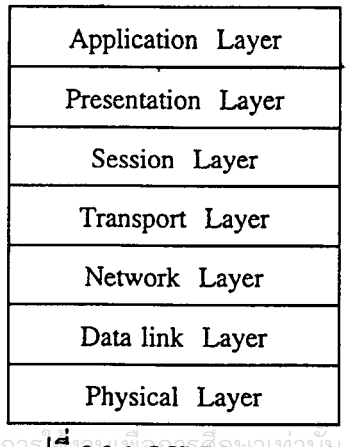
int msgrcv(int msqid , struct msgbuf *ptr , int length , long msgtype , int flag); เป็นฟังก์ชั่นที่เรียกใช้เมื่อต้องการอ่านเมสเซสขึ้นมาจากคิว ซึ่งอาร์กิวเมนต์ msgtype จะถูกกำหนดไว้เพื่อ

- ถ้า msgtype เท่ากับศูนย์ หมายถึง จะทำการอ่านเมสเซสแรกที่อยู่ในคิว
- ถ้า msgtype มากกว่าศูนย์ หมายถึง จะทำการอ่านเมสเซสแรกที่มีค่า type เท่ากับค่า msgtype ออกมา
- ถ้า msgtype น้อยกว่าศูนย์ หมายถึง จะทำการอ่านเมสเซสที่มีค่า type น้อยที่สุดหรือเท่ากับ absolute ของ msgtype ออกมา

int msgclt(int msqid , int cmd , struct msqid_ds *buff); เป็นฟังก์ชั่นที่เรียกใช้ในการควบคุมการทำงานต่างของเมสเซสคิว เช่น ลบ เมสเซสออกจากคิว

2.2 OSI Model

International Standards Organization (ISO) ได้กำหนดแบบจำลองการทำงานของระบบเครือข่ายคอมพิวเตอร์ (Computer networking model) เรียกว่า open system interconnection model (OSI Model) ได้เป็น 7 ชั้น ดังนี้



รูปที่ 2.2.1 OSI 7 layer model

1. Physical layer ทำหน้าที่ในการสื่อสารทางกายภาพ ในระดับสัญญาณทางไฟฟ้า วัสดุตัวนำที่ใช้ในการสื่อสาร การต่อเชื่อมทางกายภาพ มาตรฐานในระดับนี้ ได้แก่ RS-232, DIX-Ethernet(IEEE 802.3), Token-ring (IEEE 802.5) เป็นต้น อุปกรณ์ในเครือข่ายที่ทำงานในชั้นนี้เรียกว่า ตัวทวนสัญญาณ (Repeater)

2. Data link layer ทำหน้าที่ควบคุมการสื่อสารแบบจุดต่อจุดที่ติดกัน (Point to Point) ให้สามารถได้รับข้อมูลได้อย่างถูกต้อง ปราศจากข้อมูลที่ผิดพลาดในโหนดที่ติดกัน มาตรฐานในระดับนี้ ได้แก่ HDLC, PPP, SLIP, Ethernet, Token-ring เป็นต้น อุปกรณ์ในเครือข่ายที่ทำงานในชั้นนี้ เรียกว่าบริด (Bridge)

3. Network layer ทำหน้าที่ควบคุมการสื่อสารระหว่างต้นทางกับปลายทาง (End to End) ซึ่งในระหว่างทางจะมีเครือข่ายอยู่หรือไม่ก็ได้ รวมทั้งการหาเส้นทางที่เหมาะสมในการเดินทางของข้อมูลจากต้นทางไปยังปลายทาง (Routing) มาตรฐานในระดับนี้ ได้แก่ X.25, IP, SPX เป็นต้น อุปกรณ์ในเครือข่ายที่ทำงานในชั้นนี้ เรียกว่าเราเตอร์ (Router)

4. Transport layer ทำหน้าที่ควบคุมการสื่อสารระหว่างต้นทางและปลายทางให้สามารถได้รับข้อมูลที่ถูกต้องปราศจากข้อผิดพลาด (Error free) มาตรฐานในระดับนี้ ได้แก่ TCP, IPX เป็นต้น

5. Session layer ทำหน้าที่ควบคุมการจัดการจราจรในการสื่อสาร เช่น การหยุดส่งข้อมูลชั่วคราวเมื่อฝ่ายรับรับข้อมูลไม่ทัน หรือประมวลผลข้อมูลนั้นยังไม่เสร็จ เป็นต้น ซึ่งโดยปรกติจะจัดการโดยโปรแกรมประยุกต์เอง

6. Presentation layer ทำหน้าที่ควบคุมการแสดงผลข้อมูล การแทนค่าข้อมูล การให้ความหมายของข้อมูล เช่น จะให้ข้อมูลชุดนี้แทนรหัส ASCII หรือ EBCDIC, การตีความกลุ่มของข้อมูลว่าเป็น เลขจำนวนเต็ม หรือ ตัวอักษร หรือ จำนวนจริง เป็นต้น ซึ่งโดยปรกติจะจัดการโดยโปรแกรมประยุกต์เอง

7. Application layer เป็นระดับโปรแกรมประยุกต์ เช่น การส่งข้อความร้องขอข้อมูล การส่งข้อมูลตอบกลับ เป็นต้น ตัวอย่างเช่น FTP, SNMP, TELNET เป็นต้น

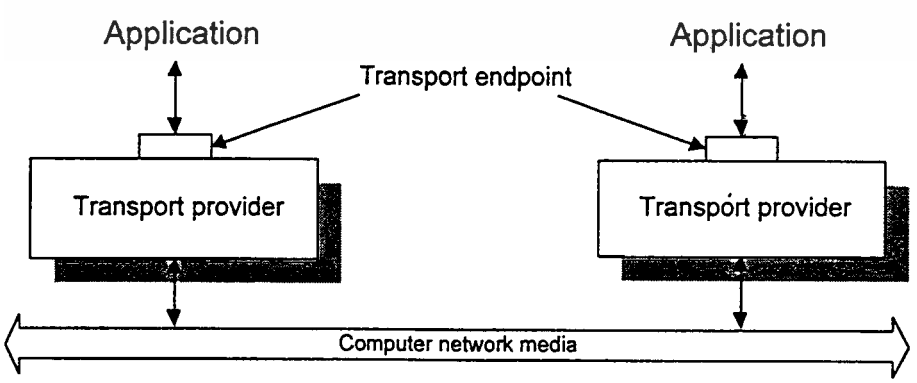
จะเห็นว่าสามารถแบ่งแบบจำลองออกได้เป็น 2 ส่วนใหญ่ๆ ตามลักษณะการใช้งานจริงๆ คือ 1. ส่วนที่รับผิดชอบโดยระบบปฏิบัติการ (O.S.) คือ ชั้นที่ 1-4 จะเรียกว่าผู้ให้บริการขนส่ง (Transport provider) และ

2. ส่วนที่อยู่ในโปรแกรมประยุกต์ คือชั้นที่ 5-7 จะเรียกว่า โปรแกรมประยุกต์ (Application program)

ดังนั้นในการสร้างโปรแกรมประยุกต์ที่จะสามารถติดต่อสื่อสารผ่านเครือข่ายได้ จะต้องเรียกใช้บริการของผู้ให้บริการขนส่ง เหมือนการเขียนโปรแกรมบนดอส จะเรียกใช้บริการของดอสในการติดต่อกับดิสก์ โดยไม่ต้องเขียนโปรแกรมติดต่อกับดิสก์โดยตรงให้ยุ่งยากและผูกติดกับดิสก์แบบนั้น หรือการเขียนโปรแกรมบนวินโดว์ ก็จะเรียกใช้บริการของวินโดว์ ซึ่งลักษณะการเรียกใช้บริการแบบนี้จะเรียกว่าเป็นการติดต่อผ่าน API (Application Program Interface) ในระบบเครือข่ายก็เช่นกันที่จะต้องมี API เช่นในระบบยูนิกส์ที่เป็นที่นิยมมีอยู่ 2 วิธี คือ BSD Sockets และ System V TLI ในไมโครซอร์ฟวินโดว์คือ Windows sockets เป็นต้น

2.3 การพัฒนาโปรแกรมผ่าน TCP/IP

ชุดโปรโตคอล TCP/IP (TCP/IP Protocol suite) เป็นโปรโตคอลที่มีในระบบปฏิบัติการแบบยูนิกส์ (UNIX Operating system) และกำลังแพร่หลายไปยังระบบปฏิบัติการอื่นๆ เช่น ดอส วินโดว์ เป็นต้น (รายละเอียดของ TCP/IP อ่านได้จาก [5] Douglas E. Comer, [6] Douglas E. Comer &David L. Stevents , [7] Douglas E. Comer&David L. Stevents) ดังนั้นในการเขียนโปรแกรมติดต่อสื่อสารผ่านระบบเครือข่ายคอมพิวเตอร์โดยใช้ชุดโปรโตคอล TCP/IP จึงเป็นแนวทางที่น่าจะดีที่สุด จากที่ได้กล่าวมาแล้วการเขียนโปรแกรมบนเครือข่ายก็จะต้องมีการเรียกใช้ผ่าน API (Application Program Interface) และปัจจุบันนี้มีมาตรฐานที่ใช้ในการเขียนโปรแกรมอยู่ 2 แบบใหญ่ๆ คือ BSD Sockets และ TLI (Transport Layer Interface) ที่จะใช้ในการให้โปรแกรมประยุกต์ติดต่อกับจุดติดต่อ (Transport endpoint)



รูปที่ 2.3.1 แสดงจุดติดต่อ (Transport endpoint)

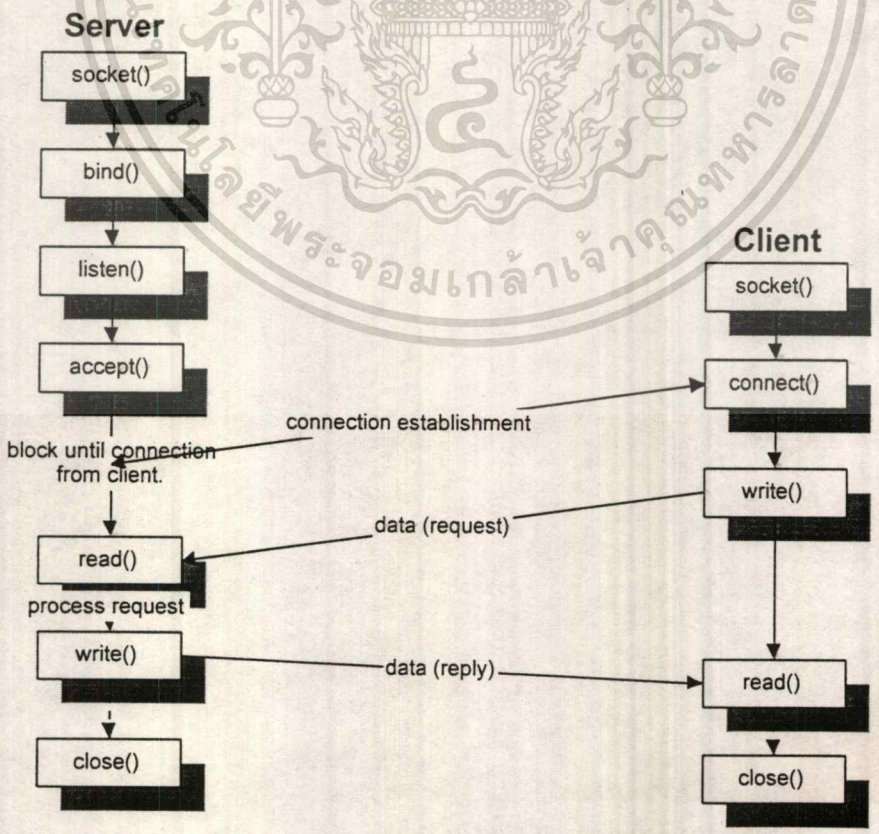
2.3.1 Sockets

Sockets หรือ **Berkeley sockets** ถือกำเนิดมาจากระบบปฏิบัติการยูนิกส์สาย BSD (Berkeley Software Distribution) ของมหาวิทยาลัยแห่งแคริฟอเนียที่เบอร์กเลย์ Sockets เป็นรูปแบบการติดต่อระหว่างโปรแกรมประยุกต์กับผู้ให้บริการขนส่ง (Transport provider) มีลักษณะเด่นคือ ชุดคำสั่งจะอยู่ในแก่นของระบบปฏิบัติการ (Kernel) และใช้คำสั่งเหมือนการจัดการกับแฟ้มข้อมูลทั่วไป เช่น close(), read(), write() เป็นต้น ทำให้ง่ายต่อการทำความเข้าใจ สามารถติดต่อกับผู้ให้บริการขนส่งที่เป็นโปรโตคอลแบบต่างๆ ได้ 3 รูปแบบ คือ

- Unix domain
- Xerox NS domain
- Internet domain (TCP/IP)

ในที่นี้จะกล่าวถึงเฉพาะแบบที่ 3 คือ Internet domain เท่านั้น (รายละเอียดใน [1] W. Richard Stevens)

การใช้ Sockets ในการติดต่อสื่อสารผ่าน TCP/IP จะแบ่งได้ 2 รูปแบบ คือ Connection-oriented protocol และ Connectionless protocol ในรูปจะแสดงการติดต่อสื่อสารโดยแบ่งตามเวลา



รูปที่ 2.3.2 แสดงการทำงานในลักษณะ connection-oriented protocol

การทำงานในโหมด connection-oriented นั้นเปรียบได้กับการโทรศัพท์ จะต้องมีการสร้างการติดต่อขึ้นมาก่อน เหมือนการหมุนโทรศัพท์ให้ติดก่อนจึงจะคุยได้ อธิบายได้ดังนี้

1. สร้างจุดติดต่อ (Transport endpoint) หรือ ต้องมีโทรศัพท์ก่อน โดยเรียก system call socket() จะได้ file descriptor ออกมา ใช้แทนจุดติดต่อต่อไป จุดติดต่อนี้จะต้องมีทั้งสองข้าง

2. ทางผู้รับ (server) จะต้องมีที่อยู่ (Address) หรือเบอร์โทรศัพท์ก่อน การกำหนดที่อยู่ให้กับจุดติดต่อนี้ทำได้โดยเรียก bind() ส่วนด้านผู้ส่ง (Client) ไม่จำเป็นต้องรู้ (แต่ต้องมี) โดยระบบจะกำหนดหมายเลขที่ว่างให้โดยอัตโนมัติ

3. ผู้รับต้องบอกว่าจะให้มีจำนวน ผู้ส่งรอได้กี่คน โดยเรียก listen()

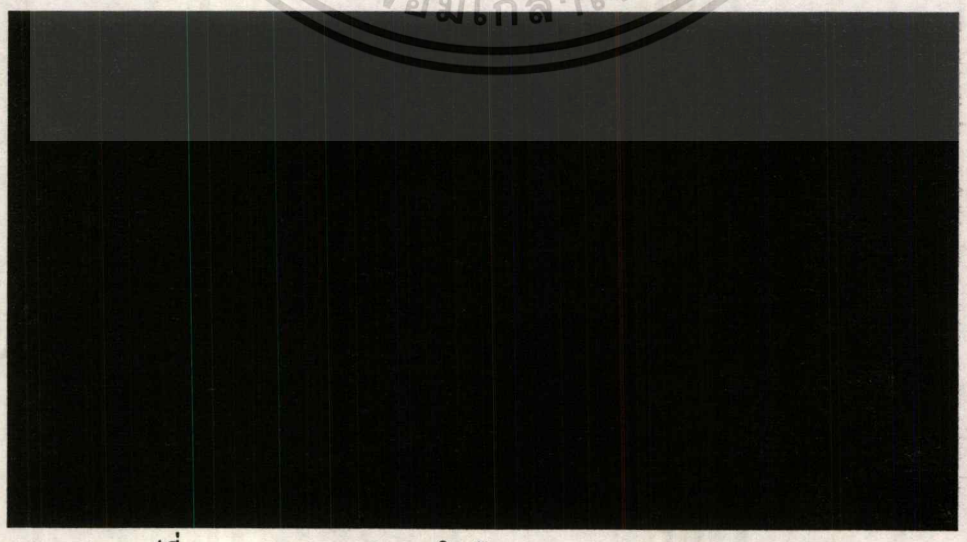
4. ผู้รับต้องรอรับ โดยเรียก accept() ซึ่งจะรอจนกระทั่งมีสัญญาณเรียกเข้ามา

5. ทางผู้ส่งก็ต้องมีการสร้างจุดติดต่อเหมือนทางผู้รับ โดยเรียก socket() เช่นกัน

6. ผู้ส่งต้องส่งสัญญาณขอติดต่อ เปรียบได้กับการหมุนโทรศัพท์ไปหาผู้รับตามเบอร์ที่รู้แล้ว โดยเรียก connect()

ขณะที่เกิดการติดต่อกัน accept() จะสร้างจุดติดต่อขึ้นอีกและจะติดต่อกับผู้ส่งทางจุดติดต่อนี้ทำให้จุดติดต่อเดิมว่างอีก ให้สายอื่นสามารถติดต่อเข้ามาได้อีก เปรียบได้กับโอเปอเรเตอร์รับที่สาย 1 แล้วโอนไปคุยกันที่สายอื่น ทำให้สาย 1 ว่างสามารถรับโทรศัพท์รายอื่นได้อีก จะเห็นได้ว่าผู้รับจะทำงานได้ 2 แบบคือ Iterative server ซึ่งจะรับได้ที่ละ 1 client และ Concurrent server ซึ่งจะรับที่ละหลายสาย (รายละเอียดใน [7] Douglas E. Comer&David L. Stevents)

เมื่อติดต่อกันได้แล้วก็สามารถรับส่งข้อมูลถึงกันได้โดยใช้ read(), write(), send(), recv() เมื่อต้องการยกเลิกการติดต่อก็เรียกใช้ close() หรือ shutdown()



รูปที่ 2.3.3 แสดงการทำงานในลักษณะ connectionless protocol

การทำงานในโหมด connectless นั้นเปรียบได้กับการส่งจดหมาย ไม่ต้องมีการสร้างการติดต่อขึ้นมาก่อน เหมือน connection-oriented อธิบายได้ดังนี้

1. สร้างจุดติดต่อ (Transport endpoint) หรือ ต้องมีกล่องจดหมายก่อน โดยเรียก system call socket() จะได้ file descriptor ออกมา ใช้แทนจุดติดต่อต่อไป จุดติดต่อนี้จะต้องมีทั้งสองข้าง

2. ทั้งผู้รับ (server) และผู้ส่ง (client) จะต้องมีที่อยู่ที่รู้กันทั้งสองฝ่าย (well known address) การกำหนดที่อยู่ให้กับจุดติดต่อนี้ทำได้โดยเรียก bind()

เมื่อมีที่อยู่แล้วก็สามารถรับส่งข้อมูลถึงกันได้โดยใช้ sendto(), recvfrom() การสื่อสารแบบนี้ไม่มีการสร้างการติดต่อขึ้นมาจึงไม่ต้องมีการยกเลิก สามารถเลิกได้โดยไม่ต้องทำอะไร แต่จะมีข้อมูลเข้ามาได้ ถ้าจะไม่ให้มีข้อมูลเข้ามาต้องทำลายตู้จดหมายหรือจุดติดต่อก่อนโดยเรียก close()

สรุปคำสั่งพื้นฐานใน Sockets

1. socket()

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int family, int type, int protocol);
```

family : ชนิดของ socket เช่น AF_INET

type : ชนิดของโปรโตคอล เช่น SOCK_STREAM (TCP), SOCK_DGRAM (UDP)

protocol : ให้มีค่าเป็น 0

return : file descriptor ของจุดติดต่อ จะใช้ต่อไปทุก system call

2. bind()

```
#include <sys/types.h>
#include <sys/socket.h>
int  bind(int fd, struct sockaddr_in *addr, int addrlen);
```

fd : จุดติดต่อที่ได้จาก socket()

addr : ที่อยู่มีโครงสร้างดังนี้

```
#include <netinet/in.h>
struct sockaddr_in{
        short           sin_family;
        u_short         sin_port;
        struct in_addr  sin_addr;
        char            sin_zero[8];
};
```

```
struct in_addr{
        u_long          s_addr;
    };
```

addrlen : ความยาวของ *addr*

return : < 0 ถ้า error

3. connect()

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, struct sockaddr_in *server_ad, int addrlen);
```

sockfd : จุดติดต่อที่ได้จาก socket() ถ้ายังไม่มีที่อยู่ (ยังไม่ได้ bind()) ระบบจะจัดการเลือกให้โดย อัตโนมัติ

server_ad : ที่อยู่ของผู้รับ

addrlen : ความยาวของ *server_ad*

return : < 0 ถ้า error

4. listen()

```
int listen(int sockfd, int backlog);
```

sockfd : จุดติดต่อที่ได้จาก socket()

backlog : จำนวนผู้ส่งที่จะให้รอขณะที่กำลังรับการติดต่อจากอีกที่

return : < 0 ถ้า error

5. accept()

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr_in *client_ad, int addrlen);
```

sockfd : จุดติดต่อที่ได้จาก socket()

client_ad : ที่อยู่ของผู้ส่งจะเป็นค่าที่ ส่งกลับจากการเรียก

addrlen : ความยาวของ *client_ad* จะเป็นค่าที่ ส่งกลับจากการเรียก

return : จุดติดต่อใหม่ ใช้ในการ read(), write() ต่อไป

6. read()

int read(int *sockfd*, char **buffer*, int *nread*);

sockfd : จุดที่จะอ่านเข้ามา

buffer : ที่เก็บข้อมูล

nread : จำนวนไบท์ที่ต้องการอ่าน

return : จำนวนที่อ่านจริง

7. write()

int write(int *sockfd*, char **buffer*, int *nwrite*);

sockfd : จุดที่จะเขียนออกไป

buffer : ที่เก็บข้อมูล

nwrite : จำนวนไบท์ที่ต้องการเขียน

return : จำนวนที่เขียนจริง

8. close()

int close(int *sockfd*);

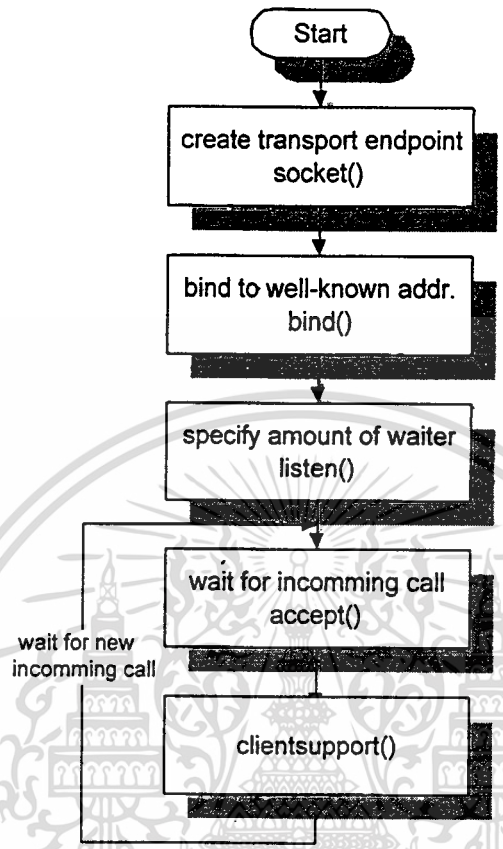
sockfd : จุดติดต่อที่ได้จาก socket()

return : < 0 ถ้า error

โครงสร้างโปรแกรม

1. Connection-oriented , Iterative server.

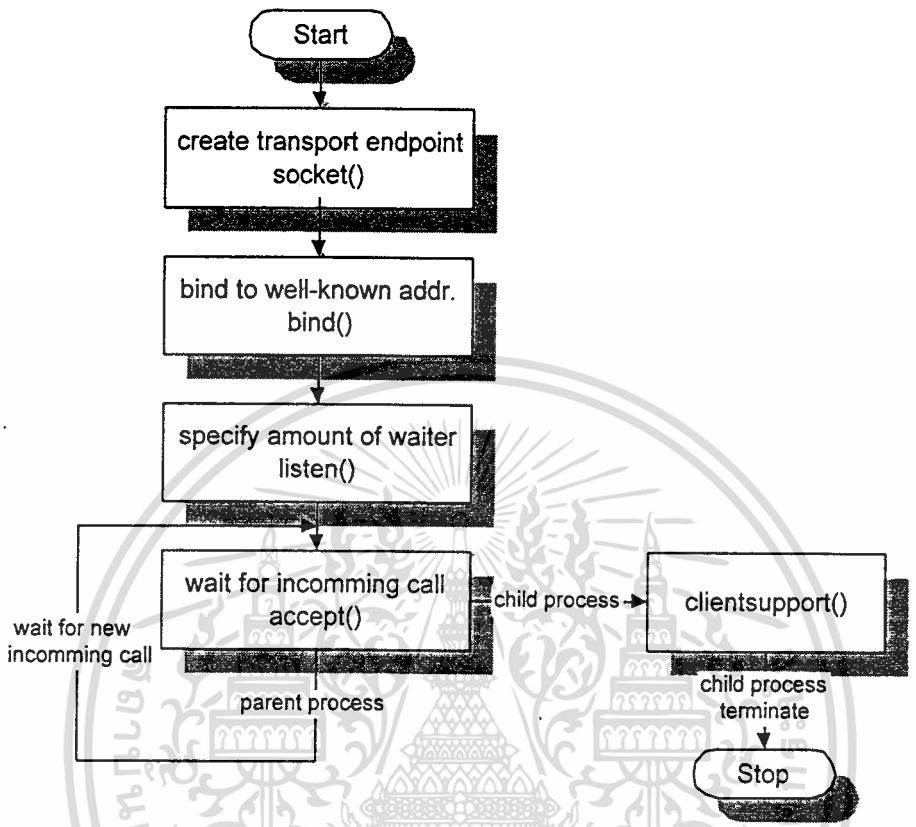
```
int    sockfd, newsockfd;
struct sockaddr_in server_ad, client_ad;
char  buffer[MAXBUF];

sockfd = socket(AF_INET, SOCK_STREAM, 0);
bzero( &server_ad, sizeof(sockaddr));              /* Fill all byte in structure to zero */
server_ad.sin_port  = htons(SERVER_PORT);     /* Convert host to network byte ordering */
server_ad.sin_addr.s_addr = htonl(SERVER_ADDR);  /* SERVER_ADDR = "161.246.4.3" */
bind(sockfd, &server_ad, sizeof(sockaddr));
listen(sockfd, 5);
for(;;){
  newsockfd = accept(sockfd,  &client_ad, sizeof(sockaddr));
  clientsupport(newsockfd);    /* do something to support client process */
  close(newsockfd);
} /* for */
```



รูปที่ 2.3.4 โฟลชาร์ตของโปรแกรม

2. Connection-oriented , Concurrent server.

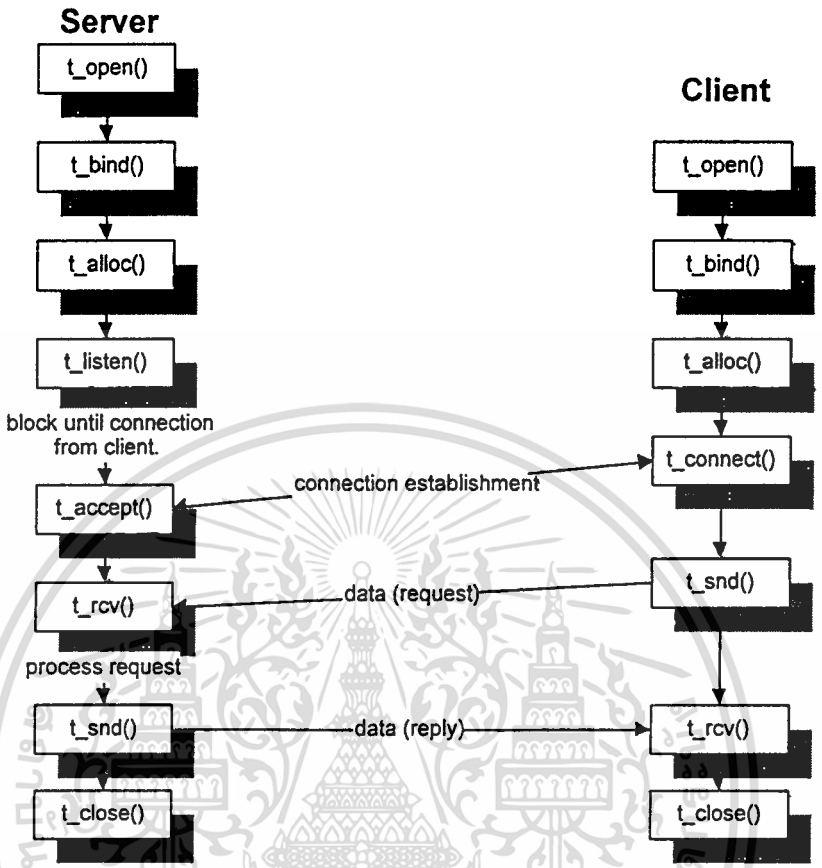
```
int   sockfd, newsockfd, childid;
struct sockaddr_in server_ad, client_ad;
char  buffer[MAXBUF];
sockfd = socket(AF_INET, SOCK_STREAM, 0);
bzero( &server_ad, sizeof(sockaddr));              /* Fill all byte in structure to zero */
server_ad.sin_port  = htons(SERVER_PORT);     /* Convert host to network byte ordering */
server_ad.sin_addr.s_addr = htonl(SERVER_ADDR);  /* SERVER_ADDR = "161.246.4.3" */
bind(sockfd, &server_ad, sizeof(sockaddr));
listen(sockfd, 5);
for(;;){
  newsockfd = accept(sockfd,  &client_ad, sizeof(sockaddr));
  childid = fork()
  if(childid == 0){      /* child process begin here */
    close(sockfd);
    clientsupport(newsockfd);    /* do something to support client process */
    exit(0);              /* child process terminate here */
  }                      /* parent process begin here */
  close(newsockfd);
} /* for */
```



รูปที่ 2.3.5 โฟลชาร์ตของโปรแกรม

2.3.2 Transport layer interface

Transport Layer Interface : TLI เป็นอีกรูปแบบหนึ่งของ API ที่ใช้สำหรับการเขียนโปรแกรมติดต่อกับผู้ให้บริการขนส่ง (Transport provider) ของระบบเครือข่าย คล้ายกับ BSD Sockets, TLI ได้กำเนิดมาจากระบบปฏิบัติการยูนิกส์ที่เป็นสาย System V ของ AT&T ซึ่งเริ่มมีใน System V release 3.0 เป็นต้นมา TLI จะมีลักษณะเด่นคือ สามารถติดต่อกับผู้ให้บริการขนส่งได้ทุกชนิด (ไม่จำเป็นต้องเป็น TCP/IP เท่านั้น) จึงสามารถเขียนโปรแกรมในลักษณะไม่ขึ้นกับชนิดของบริการขนส่ง (Transport independence) ได้ TLI ได้ถูกสร้างให้ทำงานใน user space คำสั่งจึงถูกเรียกว่า function (แทนที่จะเป็น system call เหมือน Sockets) จึงสามารถถูกสร้างขึ้นมาจากบริษัทที่สาม (Third party) ได้ ทำให้มีสินค้าให้เลือกได้มากกว่า TLI และ Sockets ก็ต่างมีข้อดีและข้อด้อยด้วยกันคือ Sockets ง่ายต่อการทำความเข้าใจและการเขียนโปรแกรม ส่วน TLI จะสามารถเขียนโปรแกรมในลักษณะไม่ขึ้นกับชนิดของบริการขนส่งได้



รูปที่ 2.3.6 แสดงการทำงานในลักษณะ connection-oriented protocol (Iterative server)

TLI ทำงานใน Connection-oriented และ Connectionless ได้เช่นเดียวกันกับ Sockets การทำงานในโหมดต่างๆ จะแสดงได้ ดังนี้

การทำงานในโหมด connection-oriented นั้นเปรียบได้กับการโทรศัพท์ จะต้องมีการสร้างการติดต่อขึ้นมาก่อน เหมือนการหมุนโทรศัพท์ให้ติดก่อนจึงจะคุยได้ อธิบายได้ดังนี้

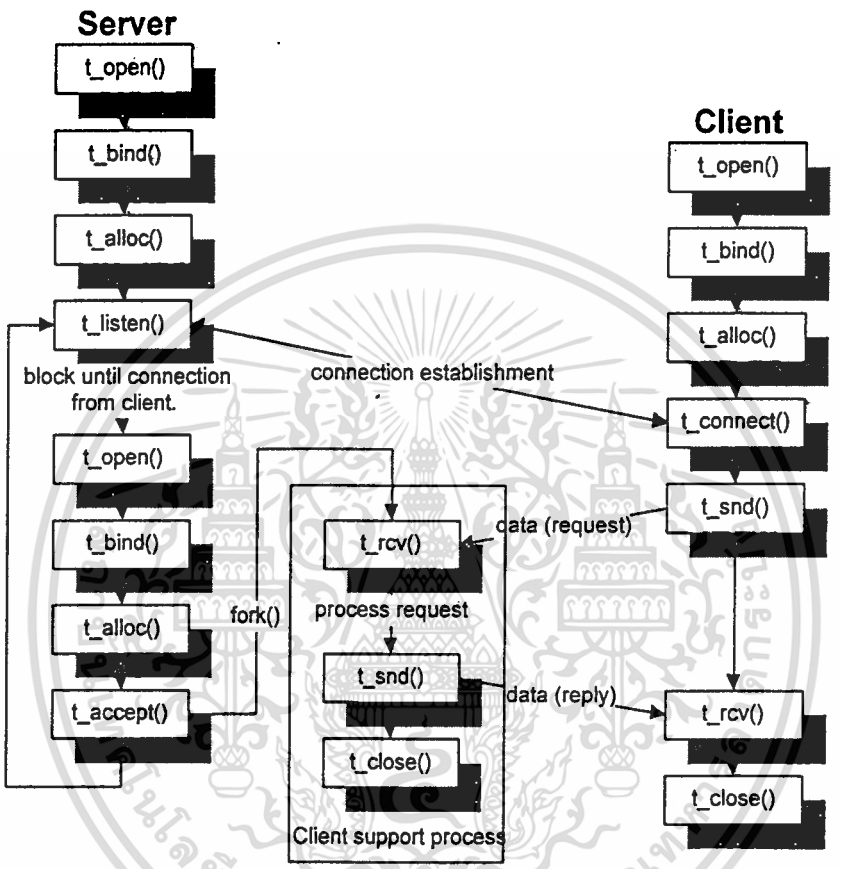
1. สร้างจุดติดต่อ (Transport endpoint) หรือ ต้องมีโทรศัพท์ก่อน โดยเรียก t_open() จะได้ file descriptor ออกมา ใช้แทนจุดติดต่อต่อไป จุดติดต่อนี้จะต้องมีทั้งสองข้าง

2. กำหนดที่อยู่ (Address) หรือเบอร์โทรศัพท์ให้กับทั้งสองข้าง การกำหนดที่อยู่ให้กับจุดติดต่อนี้ทำได้โดยเรียก t_bind() ทางด้านผู้ส่ง (Client) ไม่จำเป็นต้องเป็นที่อยู่ที่รู้กัน

3. ในการเรียกใช้ฟังก์ชั่นเกือบทั้งหมดจะมี การส่งผ่านข้อมูลระหว่างกัน โดยจะส่งเป็นตัวชี้ (Pointer) แทนที่จะส่งเป็นข้อมูลจริง ขนาดของข้อมูลที่ส่งผ่านกันนี้จะมีขนาดแตกต่างกันตามชนิดของผู้ให้บริการขนส่ง ดังนั้นเทคนิคการจองพื้นที่สำหรับรับส่งข้อมูลจึงต้องใช้การทำแบบไดนามิก (Dynamic allocation) โดยใช้ฟังก์ชั่น t_alloc() ในการจองพื้นที่ การจองพื้นที่ต้องทำทั้งสองฝ่าย

4. ฝ่ายรับต้องรอรับ โดยเรียก t_listen() ซึ่งจะรอจนกระทั่งมีสัญญาณเรียกเข้ามา

5. ฝ่ายส่งต้องส่งสัญญาณขอติดต่อ เปรียบได้กับการหมุนโทรศัพท์ไปหาผู้รับตามเบอร์ที่รู้แล้ว โดยเรียก t_connect()



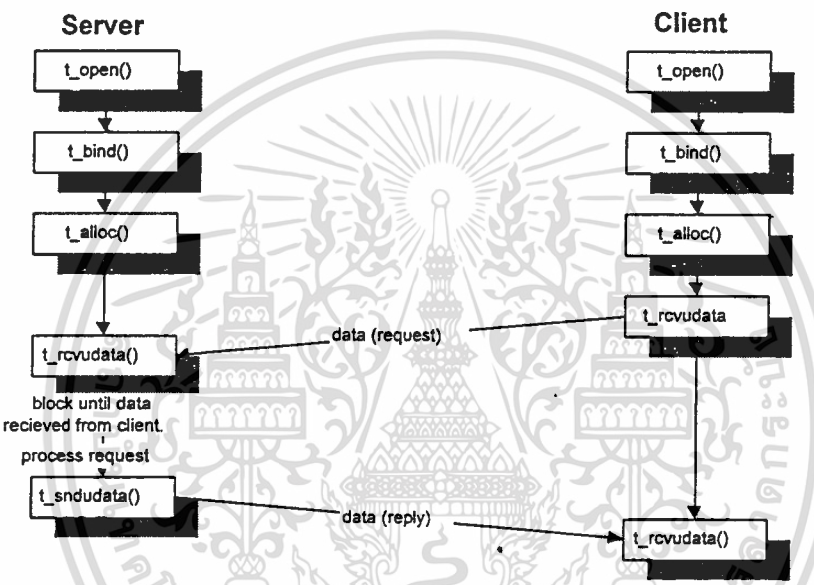
รูปที่ 2.3.7 แสดงการทำงานในลักษณะ connection-oriented protocol (Concurrent server)

6. ฝ่ายรับเมื่อมีสัญญาณขอติดต่อเข้ามา (เรียกโดย t_connect() และรอรับโดย t_listen()) t_listen() จะจบการทำงาน ฝ่ายรับต้องรับโดยเรียก t_accept()

ขณะที่เกิดการติดต่อกัน accept() จะไม่มีการสร้างจุดติดต่อขึ้นอีกเหมือนกับ sockets ดังนั้นถ้ามีการรับกับหมายเลขพอร์ท (port) เดิม ก็จะเป็นการทำงานแบบ Iterative server หรือมีการสร้างพอร์ทขึ้นใหม่และรับกับหมายเลขพอร์ทใหม่นี้ หมายเลขพอร์ทเดิมก็จะว่างให้มีการติดต่อเข้ามาได้อีกเหมือนกับ sockets ซึ่งมี 2 ลักษณะเช่นกันคือ ตัวโปรเซสเดียวกันนี้เข้าไปทำงานในส่วนที่ติดต่อกับผู้ใช้บริการ ก็จะเป็นลักษณะ Iterative server เหมือนเดิม ถ้าโปรเซสนั้นสร้างโปรเซสใหม่ (ด้วยการfork()) เพื่อทำงานในส่วนผู้ใช้บริการและโปรเซสหลักก็กลับไปรอรับการเรียกเข้ามาอีกก็จะเป็นแบบ Concurrent server เปรียบได้กับโอเปอเรเตอร์รับที่สาย 1 แล้วโอนไปคุยกันที่สายอื่นทำให้สาย 1 ว่าง

สามารถรับโทรศัพท์รายอื่นได้อีก จะเห็นได้ว่า ผู้รับจะทำงานได้ 2 แบบคือ Iterative server ซึ่งจะรับได้ที่ละ 1 ไคลแอนต์และ Concurrent server ซึ่งจะรับที่ละหลายไคลแอนต์ (รายละเอียดใน [7] Douglas E. Comer&David L. Stevents)

เมื่อติดต่อกันได้แล้วก็สามารถรับส่งข้อมูลถึงกันได้โดยใช้ t_rcv(), t_snd() เมื่อต้องการยกเลิกการติดต่อก็เรียกใช้ t_close(), t_sndrel() หรือ t_snddis โดย t_close() จะยกเลิกการติดต่อทันที แต่ t_sndrel(), t_snddis() จะต้องรอให้อีกฝ่ายหนึ่งรับโดยเรียก t_rcvrel(), t_rcvdis() ตามลำดับ



รูปที่ 2.3.8 แสดงการทำงานในลักษณะ connectionless protocol

การทำงานในโหมด connectless นั้นเปรียบได้กับการส่งจดหมาย ไม่ต้องมีการสร้างการติดต่อขึ้นมาก่อน เหมือน connection-oriented อธิบายได้ดังนี้

1. สร้างจุดติดต่อ (Transport endpoint) หรือ ต้องมีกล่องจดหมายก่อน โดยเรียก t_open() จะได้ file descriptor ออกมา ใช้แทนจุดติดต่อต่อไป จุดติดต่อนี้จะต้องมีทั้งสองข้าง

2. ทั้งผู้รับ (server) และผู้ส่ง (client) จะต้องมีที่อยู่ที่รู้กันทั้งสองฝ่าย (well known address) การกำหนดที่อยู่ให้กับจุดติดต่อนี้ทำได้โดยเรียก t_bind()

เมื่อมีที่อยู่แล้วก็สามารถรับส่งข้อมูลถึงกันได้โดยใช้ t_sndudata(), t_rcvudata() การสื่อสารแบบนี้ไม่มีการสร้างการติดต่อขึ้นมาจึงไม่ต้องมีการยกเลิก สามารถเลิกได้โดยไม่ต้องทำอะไร แต่จะมีข้อมูลเข้ามาได้ ถ้าจะไม่ให้มีข้อมูลเข้ามาต้องทำลายตู้จดหมายหรือจุดติดต่อก่อนโดยเรียก t_close()

2.3.3 Windows sockets

Windows sockets หรือ **WINSOCK** คือ รูปแบบการเขียนโปรแกรมติดต่อสื่อสารผ่านเครือข่ายคอมพิวเตอร์ที่ใช้โปรโตคอล TCP/IP บนไมโครซอร์ฟวินโดว์ ซึ่งมีหลักการและชุดคำสั่งเหมือนกับ BSD sockets บนระบบปฏิบัติการยูนิกส์ดังที่ได้กล่าวมาแล้ว แต่เนื่องจากไมโครซอร์ฟวินโดว์เป็นระบบปฏิบัติการแบบ Non-preemtive จึงทำให้จำเป็นต้องมีชุดคำสั่งที่เป็นแบบไมโครซอร์ฟวินโดว์โดยเฉพาะ ในไมโครซอร์ฟวินโดว์นั้นการเรียกใช้บริการแบบอินพุท เอ้าท์พุท (I/O operation) จะมีการรอ (blocking) จนกระทั่งการทำงานนั้นสำเร็จจึงจะคืนการทำงานให้กับระบบ ชุดคำสั่งเฉพาะของไมโครซอร์ฟวินโดว์จึงเป็นแบบไม่มีการรอ (non-blocking) แต่จะใช้ข้อความบอกว่ามีเหตุการณ์อะไรเกิดขึ้นบ้าง เช่น ขณะที่รอรับข้อมูลจะใช้คำสั่ง recv() โดยตรงไม่ได้เพราะว่าระบบจะรอจนมีข้อมูลพร้อมให้อ่านจึงจะคืนการทำงานให้ระบบ เป็นต้น ดังนั้นการเขียนโปรแกรมจึงเป็นแบบรอให้มีข้อมูลพร้อมให้อ่านจึงใช้คำสั่ง recv() เพื่ออ่านข้อมูลนั้นเข้ามา การรอนี้จะใช้การบอกแก่ระบบว่าถ้ามีเหตุการณ์ที่ต้องการเกิดขึ้นที่จุดติดต่อ (transport endpoint) ให้ส่งข้อความบอกด้วย เหตุการณ์ที่เกิดขึ้นที่จุดติดต่อมีดังนี้

- FD_READ คือ มีข้อมูลพร้อมให้อ่าน
- FD_WRITE คือ ข้อมูลถูกส่งแล้ว
- FD_OOB คือ มีข้อมูลด่วนพร้อมให้อ่าน
- FD_ACCEPT คือ มีการขอติดต่อเข้ามาที่ฝ่ายรอรับ
- FD_CONNECT คือ การขอติดต่อไปได้ตอบรับแล้วที่ฝ่ายร้องขอ
- FD_CLOSE คือ socket นั้นได้ยุติการติดต่อแล้ว

คำสั่งเฉพาะของไมโครซอร์ฟวินโดว์นั้นจะมีชื่อขึ้นต้นด้วย WSA (Windows Sockets Asynchronous) เสมอ ซึ่งมีที่สำคัญดังนี้

- WSAStartup() ใช้เรียกก่อนคำสั่งเฉพาะไมโครซอร์ฟวินโดว์อื่นๆ เพื่อตรวจสอบเวอร์ชั่นของ WINSOCK.DLL และเป็นการเริ่มต้นใช้งาน
- shutdown() ใช้เมื่อจะเลิกการติดต่อ (disconnect) แต่จุดติดต่อยังไม่ถูกยกเลิก
- closesocket() ใช้เพื่อยกเลิกจุดติดต่อหลังจากที่ใช้ shutdown() เลิกการติดต่อแล้ว

ส่วนคำสั่งอื่นๆจะใช้เช่นเดียวกับ sockets บนระบบยูนิกส์

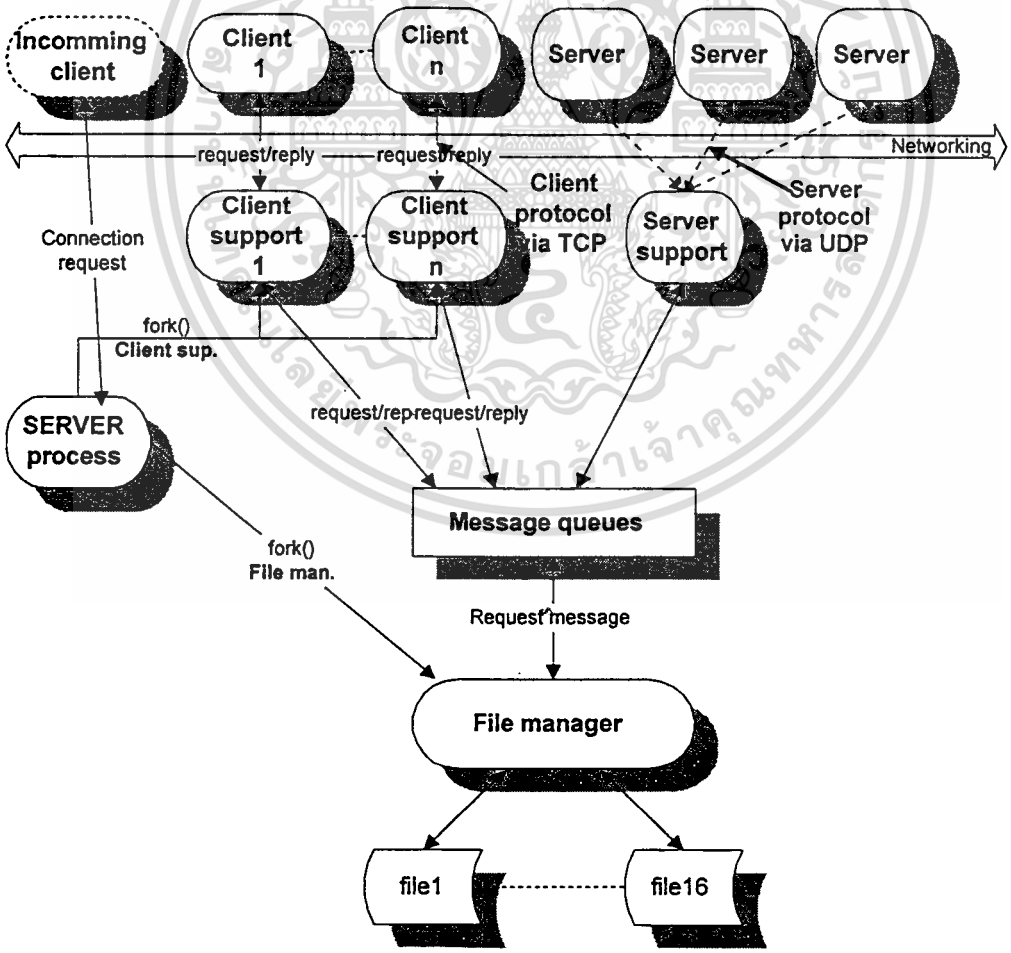
บทที่ 3

แนวคิดและการทำงาน

3.1 หลักการทำงานของระบบ

ระบบจะเป็นลักษณะผู้ให้บริการข้อมูลแบบกระจาย (Distributed data server) ที่สามารถเก็บข้อมูลกระจายไว้ในส่วนต่างๆ ของเครือข่ายคอมพิวเตอร์ที่ใช้โปรโตคอล TCP/IP ผู้ใช้บริการ (client)เมื่อติดต่อเข้าในระบบที่ผู้ให้บริการ (server) ตัวใดตัวหนึ่งแล้ว จะสามารถจัดการกับข้อมูลจากผู้ให้บริการได้ทุกตัวที่ต่ออยู่ในระบบขณะนั้นโดยไม่ต้องรู้ว่าข้อมูลจริงๆแล้วเก็บไว้ที่ใด ผู้ให้บริการจะจัดการกับข้อมูลที่อยู่ต่างเครื่องให้โดยอัตโนมัติ ระบบจะประกอบด้วยผู้ให้บริการ มีได้มากที่สุด 16 ตัวและผู้ใช้บริการ จำนวนไม่จำกัดขึ้นอยู่กับระบบปฏิบัติการ

ในแต่ละโหนดที่ทำหน้าที่เป็นผู้ให้บริการ (server) จะประกอบไปด้วยส่วนต่างๆ ดังรูป



รูปที่ 3.1.1 Single filemanager model

3.1.1 Server process

เป็นโปรเซสหลัก เมื่อมีการเรียกให้โปรแกรมทำงาน ส่วนแรกที่จะทำงานก็คือส่วนนี้เป็นส่วนแรก และกระทำสิ่งต่างๆ ดังต่อไปนี้

- ทำการเช็ค *.DB ที่มีอยู่และทำการ initial file ลง locol table

- ทำการสร้างจุดติดต่อ 2 จุด สำหรับโปรโตคอล TCP และ UDP ถ้าเกิดข้อผิดพลาดขึ้นให้ออกจากโปรแกรม

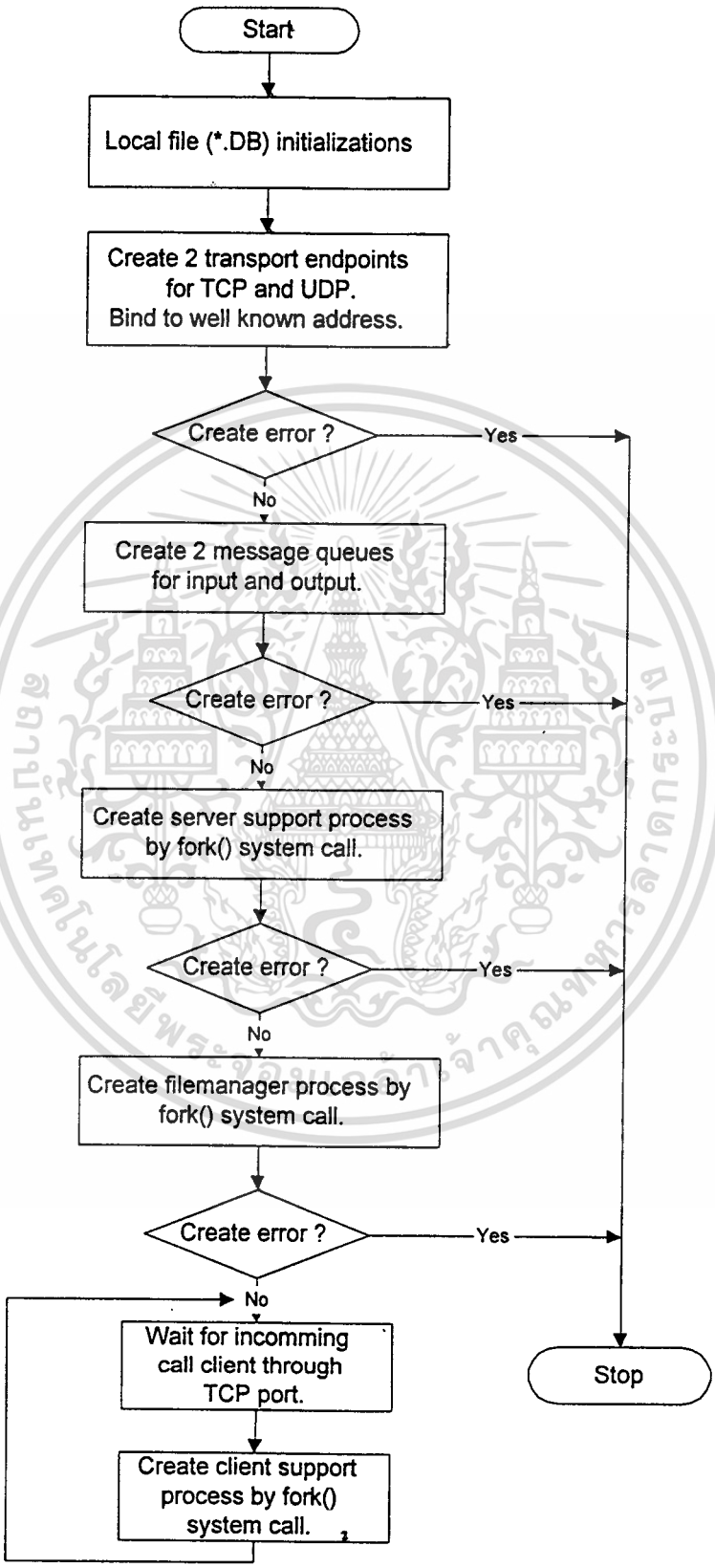
- สร้างเมสเซสคิวทั้งอินพุทและเอ๊าท์พุทเพื่อใช้ติดต่อกับ file manager ถ้าเกิดข้อผิดพลาดขึ้นให้ออกจากโปรแกรม

- สร้างโปรเซส server support เพื่อทำการติดต่อกับ server อื่นๆ ถ้าเกิดข้อผิดพลาดขึ้นให้ออกจากโปรแกรม

- สร้างโปรเซส File manager ซึ่งทำหน้าที่จัดการกับแฟ้มข้อมูลทั้งหมด ถ้าเกิดข้อผิดพลาดขึ้นให้ออกจากโปรแกรม

- รอรับการเรียกเข้ามาของผู้ใช้บริการ ซึ่งเมื่อมีเรียกเข้ามา จะทำการสร้างโปรเซสใหม่ขึ้นมาเพื่อติดต่อกับผู้ใช้บริการโดยตรง เรียกว่า client support process

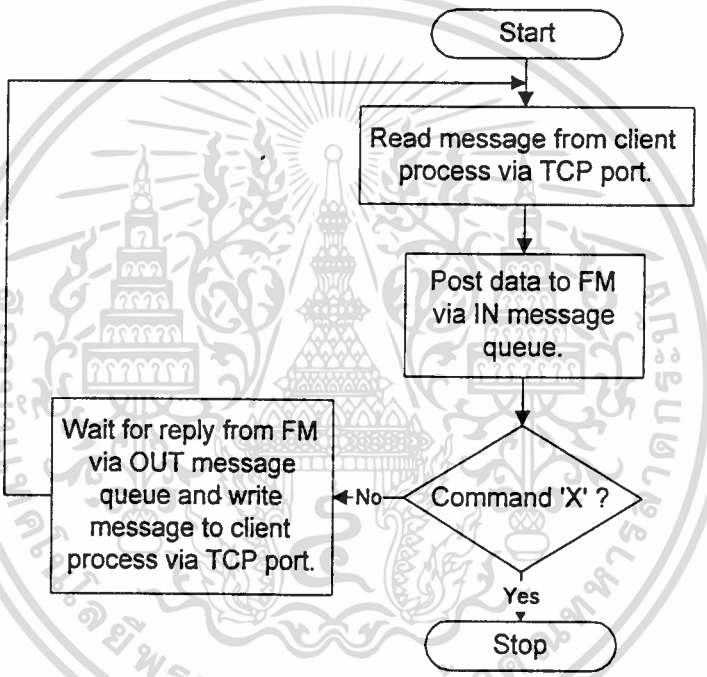
- กลับไปรอรับการเรียกเข้ามาของผู้ใช้บริการรายใหม่ต่อไป



รูปที่ 3.1.2 Flow chart แสดงการทำงานของ Server process

3.1.2 Client support process

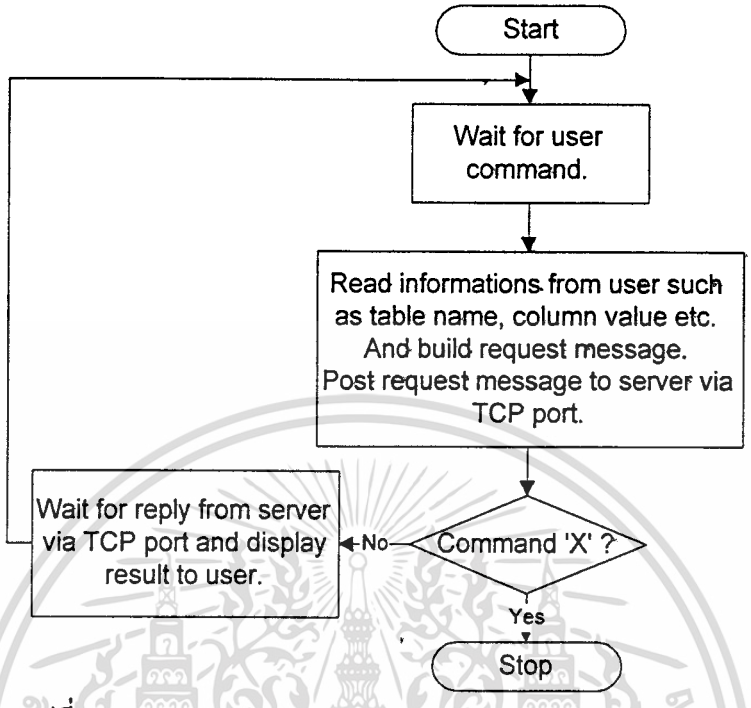
เป็นโปรเซสที่จะติดต่อสื่อสารผ่านเครือข่ายกับโปรเซสของผู้ใช้บริการ (client) เพื่อรอรับข้อความ (message) ร้องขอข้อมูลและส่งไปที่ file manager ผ่านเมสเซสคิว จากนั้นโปรเซสที่จัดการแฟ้มข้อมูล (file manager process) ก็จะหยิบเมสเซสนั้นขึ้นมาอ่านและทำการตอบสนองส่งผลลัพธ์กลับไปที่ผู้ใช้บริการ แต่ตรวจเช็ค message ที่ส่งมาจาก client เป็นคอมมานด์ X ก็จะเป็นการจบ client support process ซึ่งจะเห็นว่าเหมือนเป็นการส่งผ่านข้อมูลที่ผ่านเข้าออกไปเฉยๆ



รูปที่ 3.1.3 Flow chart แสดงการทำงานของ Client support process

3.1.3 Client process

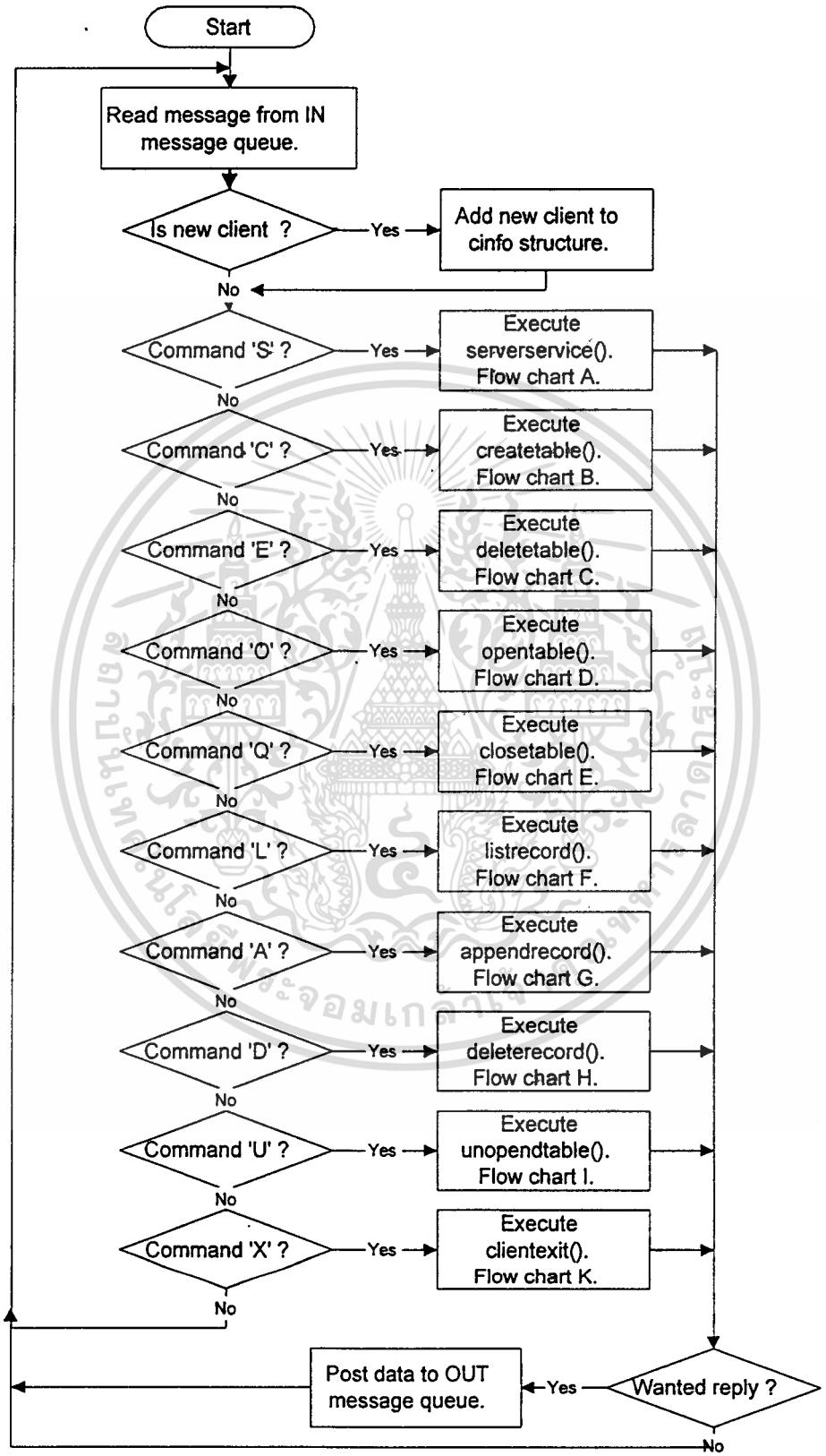
เป็นโปรเซสที่ทำการติดต่อกับผู้ใช้ซึ่งจะเรียกใช้ข้อมูลจากผู้ให้บริการ (server) สามารถเป็นโปรเซสที่ทำงานอยู่เครื่องอื่นหรือเครื่องเดียวกับผู้ให้บริการก็ได้ โดยจะติดต่อผ่านเครือข่ายเข้ามาที่ server process แล้ว server process จะสร้างโปรเซสอีกโปรเซสหนึ่งขึ้นมาทำงานเพื่อรอรับการร้องขอเรียกโปรเซสนี้ว่า Client support process โดยรูปแบบของการร้องขอจะเรียกว่า client protocol



รูปที่ 3.1.4 Flow chart แสดงการทำงานของ Client process

3.1.4 File manager

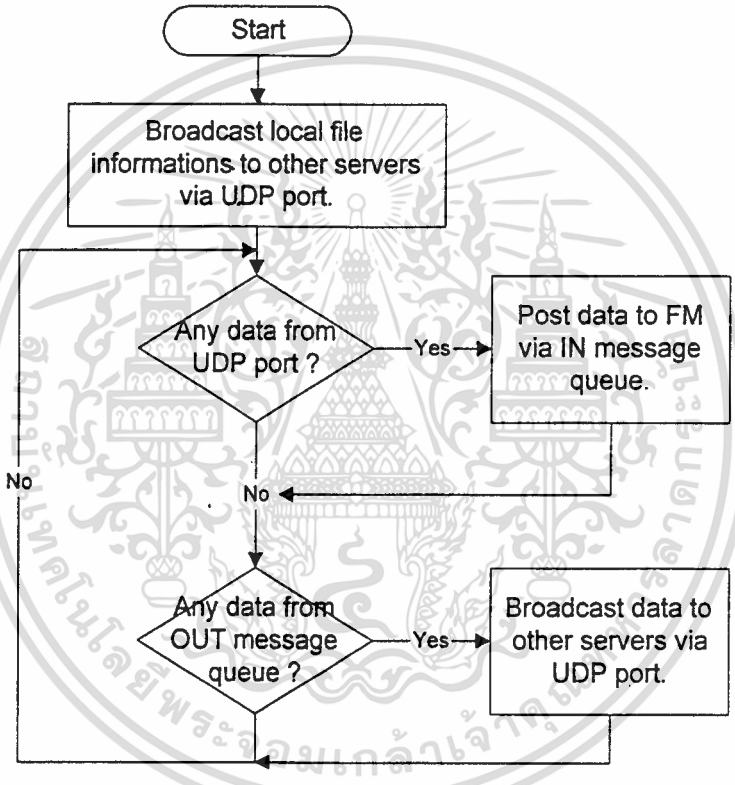
File manager : จะทำหน้าที่เกี่ยวกับการจัดการแฟ้มข้อมูลทั้งหมด ทั้งที่อยู่ต่างเครื่องและเครื่องเดียวกัน โดยจะรอรับข้อความร้องขอจากไคลแอนต์ ผ่านทางเมสเซสคิวและปฏิบัติตามการร้องขอนั้นแล้วจึงตอบกลับไปที่ไคลแอนต์ ในกรณีที่ข้อมูลที่ร้องขอไม่ได้อยู่ในเครื่องเดียวกันนี้ ก็จะมีการส่งข้อความเพื่อร้องขอจากเครื่องอื่นๆในระบบ โดยข้อความและขั้นตอนในการติดต่อกันระหว่าง File manager นี้เรียกว่า **server protocol**



รูปที่ 3.1.5 Flow chart แสดงการทำงานของ File manager

3.1.5 Server support process

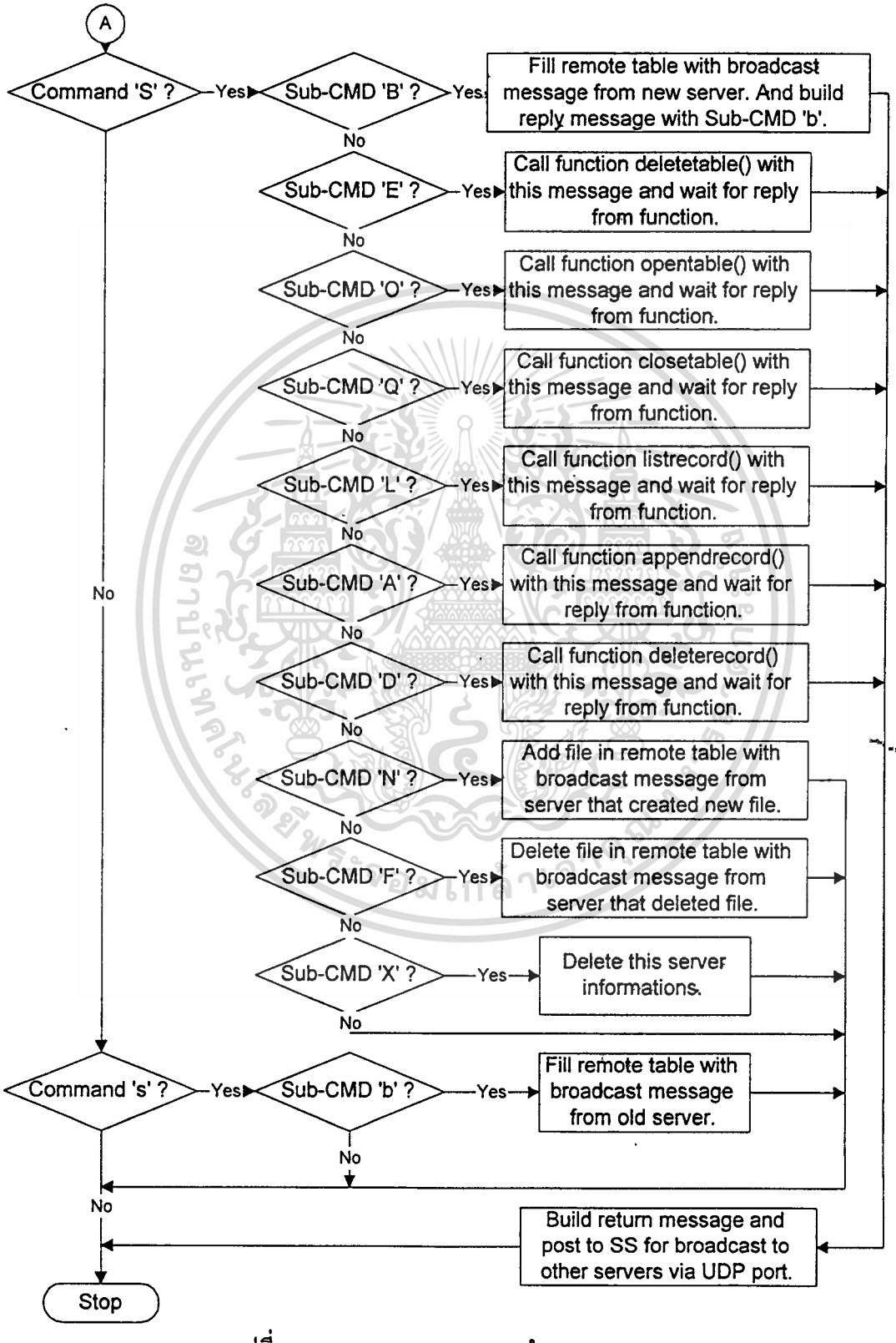
เป็นโปรเซสที่มีไว้เพื่อรองรับในการติดต่อกับผู้ให้บริการตัวอื่นๆ โดยเริ่มแรกจะทำการบอกผู้ให้บริการตัวอื่นๆถึงไฟล์ต่างๆที่ตนเองมีอยู่ แล้วก็ทำการเช็คดูว่าผู้ให้บริการตัวอื่นๆส่งข้อความอะไรเข้ามาหรือเปล่า ถ้ามีก็ส่งต่อไปยังโปรเซสที่จัดการไฟล์ผ่านเมสเซสคิว จากนั้นก็เช็คดูว่ามีเมสเซสจากเอ๊าท์พุทเมสเซสคิวหรือเปล่า ถ้ามีก็ส่งข้อความไปให้ผู้ให้บริการตัวอื่นๆ



รูปที่ 3.1.6 flowchart แสดงการทำงานของ server support process

3.1.6 การทำงานของ File manager ต่อคำสั่งต่างๆ

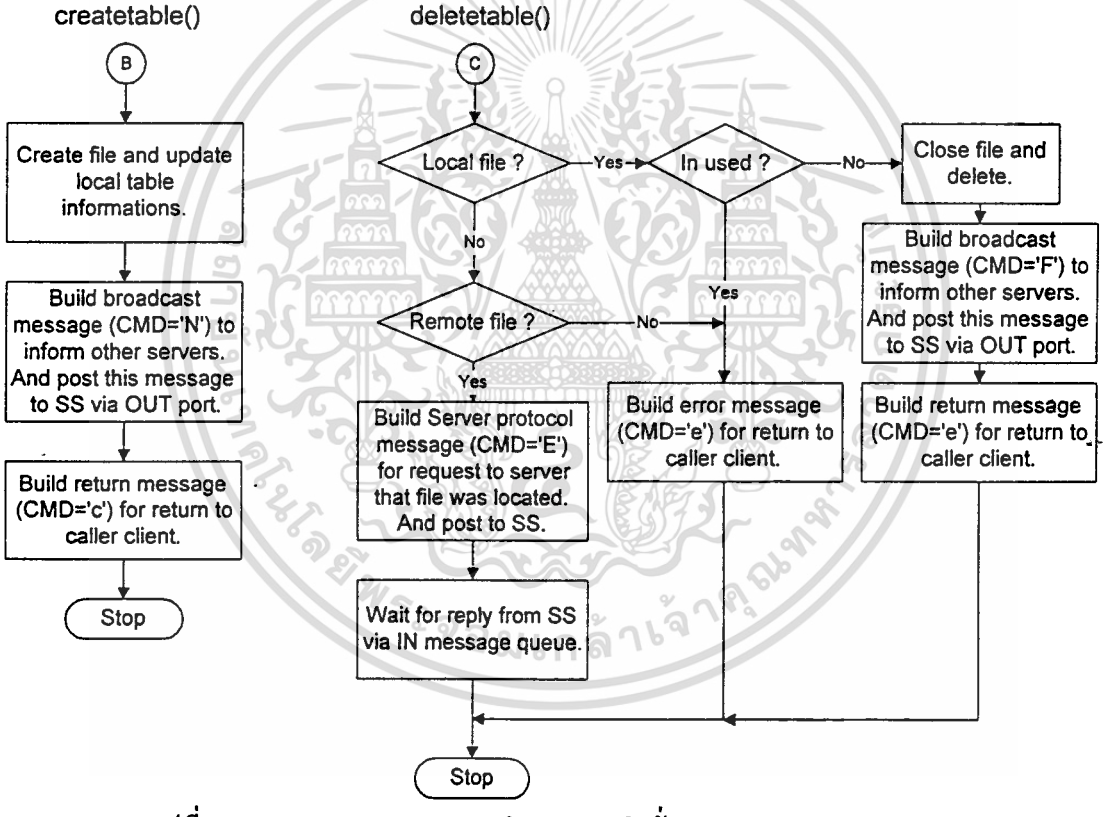
Server service เป็นการขอการบริการจากเซอร์เวอร์ตัวอื่นๆ โดยตรวจคำสั่งถ้า command = S หมายถึง เป็นคำสั่งการร้องขอ (request) ภายในคำสั่งนี้จะมีคำสั่งย่อยๆต่อไปอีก โดยที่เซอร์เวอร์ตัวให้บริการจะทำตามการร้องขอและเมื่อทำสำเร็จก็จะทำการกระจายข้อมูลไปให้เซอร์เวอร์ตัวที่ร้องขอมา ซึ่งจะได้กล่าวต่อไปในส่วนของ server protocol แต่ถ้า command = s หมายถึง เป็นการกระจายข้อมูลข่าวสารจากเซอร์เวอร์ตัวอื่นๆ ดังนั้นเซอร์เวอร์ตัวที่ได้รับข้อมูลก็จะทำการเก็บข้อมูลเกี่ยวกับเซอร์เวอร์ตัวอื่นๆไว้



รูปที่ 3.1.7 flowchart แสดงการทำงานของ Server service

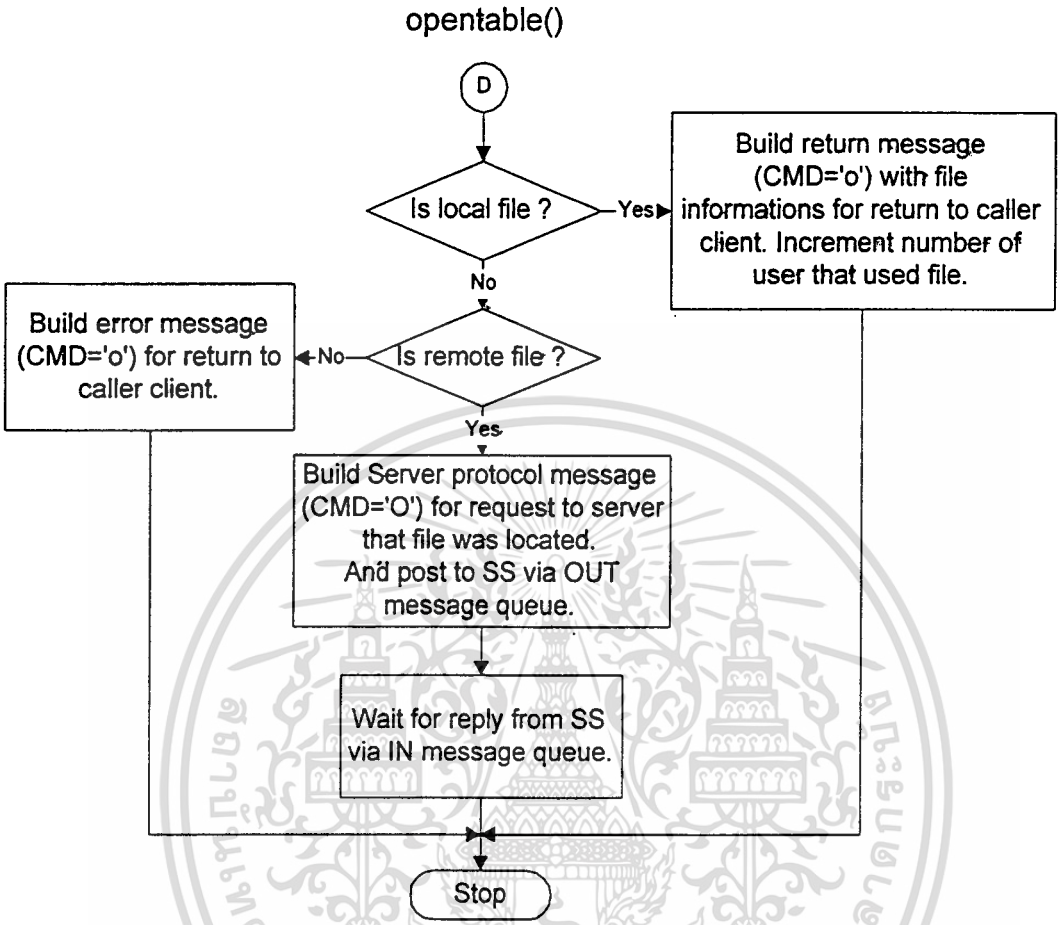
Creat table เริ่มต้นโดยการสร้างไฟล์ขึ้นมาแล้วทำการ update ข้อมูลลงใน local table จากนั้นก็ทำการกระจายข้อมูลเพื่อแจ้งให้เซอร์เวอร์อื่นรู้ว่ามีไฟล์ใหม่เกิดขึ้น แจ้งเมสเซสกับไปยังผู้ใช้ว่าสร้างตารางใหม่สำเร็จหรือเกิดข้อผิดพลาด

Delete table เริ่มก็จะต้องเช็คให้ได้ก่อนว่าตารางที่ต้องการลบเป็น local หรือ remote ถ้าเป็น local แล้วไม่มีการเปิดใช้งานอยู่ก็สามารถลบได้เลยแล้วก็ส่งข้อความให้เซอร์เวอร์อื่นรู้ ต่อจากนั้นก็ส่งข้อความบอกผู้ใช้ว่าลบได้หรือไม่ได้ แต่ถ้าไฟล์นั้นเปิดใช้งานอยู่ก็ไม่สามารถลบได้ ถ้าเป็น remote ไฟล์ก็จะต้องส่งเมสเซสนี้ไปให้เซอร์เวอร์อื่นๆที่มีไฟล์นั้นอยู่เป็นตัวจัดการ



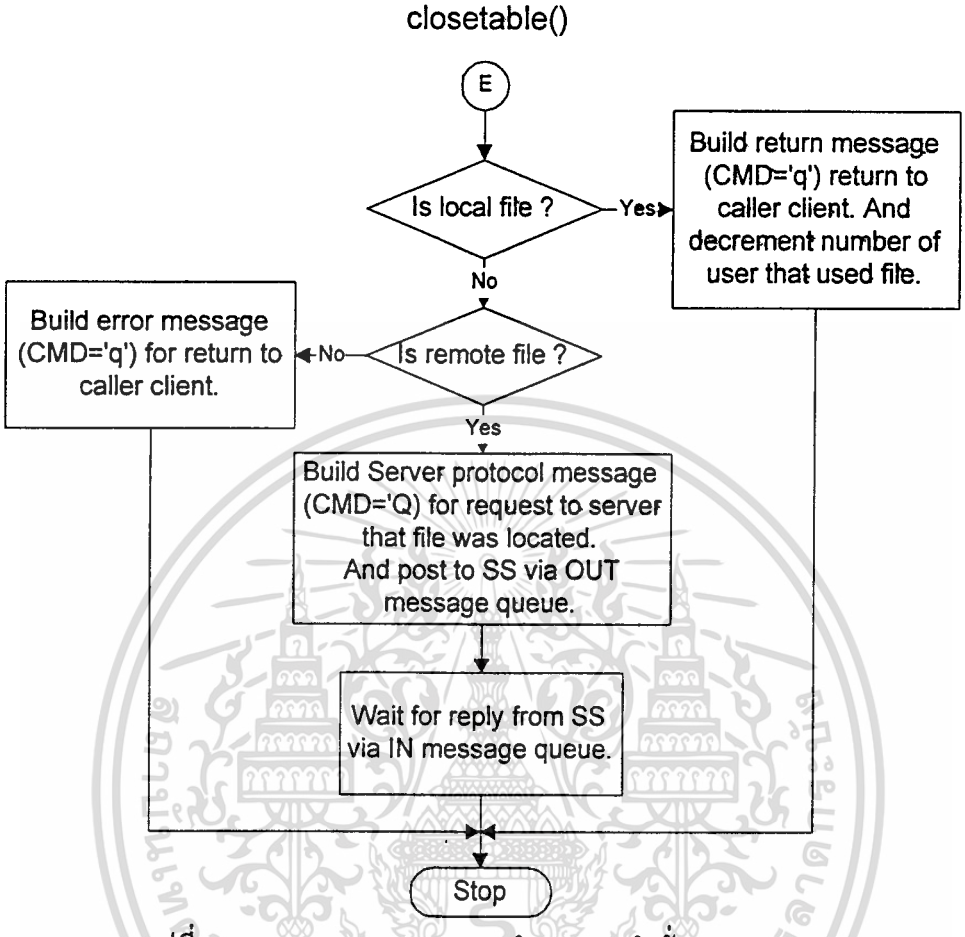
รูปที่ 3.1.8 flowchart แสดงการทำงานของคำสั่ง Creat และ Delete table

Open table จะต้องเช็คให้ได้ก่อนว่าไฟล์ที่ต้องการ open นั้นเป็น local หรือ remote ถ้าเป็น local ก็สามารถทำการเปิดไฟล์ขึ้นใช้งานได้เลยและเพิ่มจำนวนผู้ใช้งานไฟล์นั้นๆขึ้นหนึ่ง แต่ถ้าไฟล์นั้นเป็น remote ก็จะต้องส่งเมสเซสนี้ไปให้เซอร์เวอร์อื่นๆที่มีไฟล์นั้นอยู่เป็นตัวจัดการ แล้วรอรับคำตอบจากโปรเซส server support แต่ถ้าไฟล์นั้นไม่มีอยู่ทั้งใน local และ remote ก็ไม่สามารถเปิดไฟล์นั้นได้



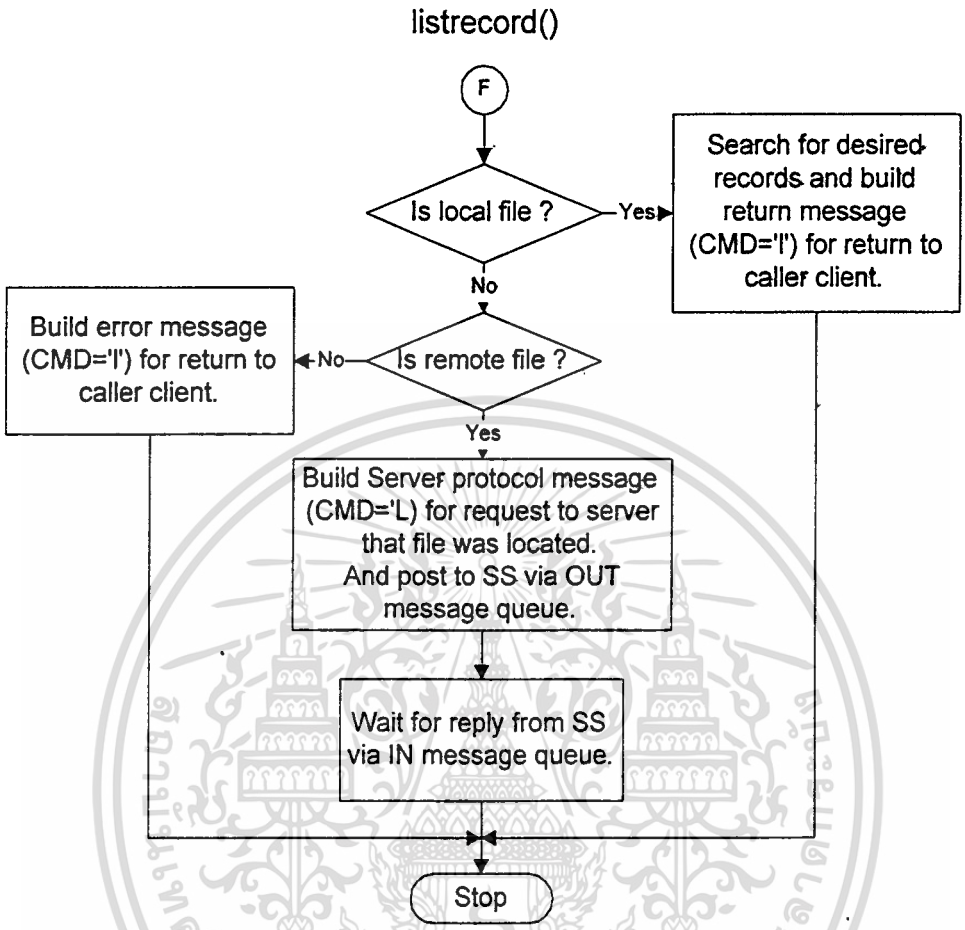
รูปที่ 3.1.9 flowchart แสดงการทำงานของคำสั่ง open table

Close table จะต้องเช็คให้ได้ก่อนว่าไฟล์ที่ต้องการ close นั้นเป็น local หรือ remote ถ้าเป็น local ก็สามารถทำการปิดไฟล์นั้นได้เลยถ้าจำนวนผู้ใช้ไฟล์เท่ากับหนึ่ง แต่ถ้ามากกว่าให้ลดจำนวนผู้ใช้ไฟล์ลงหนึ่ง แต่ถ้าไฟล์นั้นเป็น remote ก็จะต้องส่งเมสเซสนี้ไปให้เซอร์เวอร์อื่นๆที่มีไฟล์นั้นอยู่เป็นตัวจัดการ แล้วรอรับคำตอบจากโปรเซส server support แต่ถ้าไฟล์นั้นไม่มีอยู่ทั้งใน local และ remote ก็ไม่สามารถปิดไฟล์นั้นได้



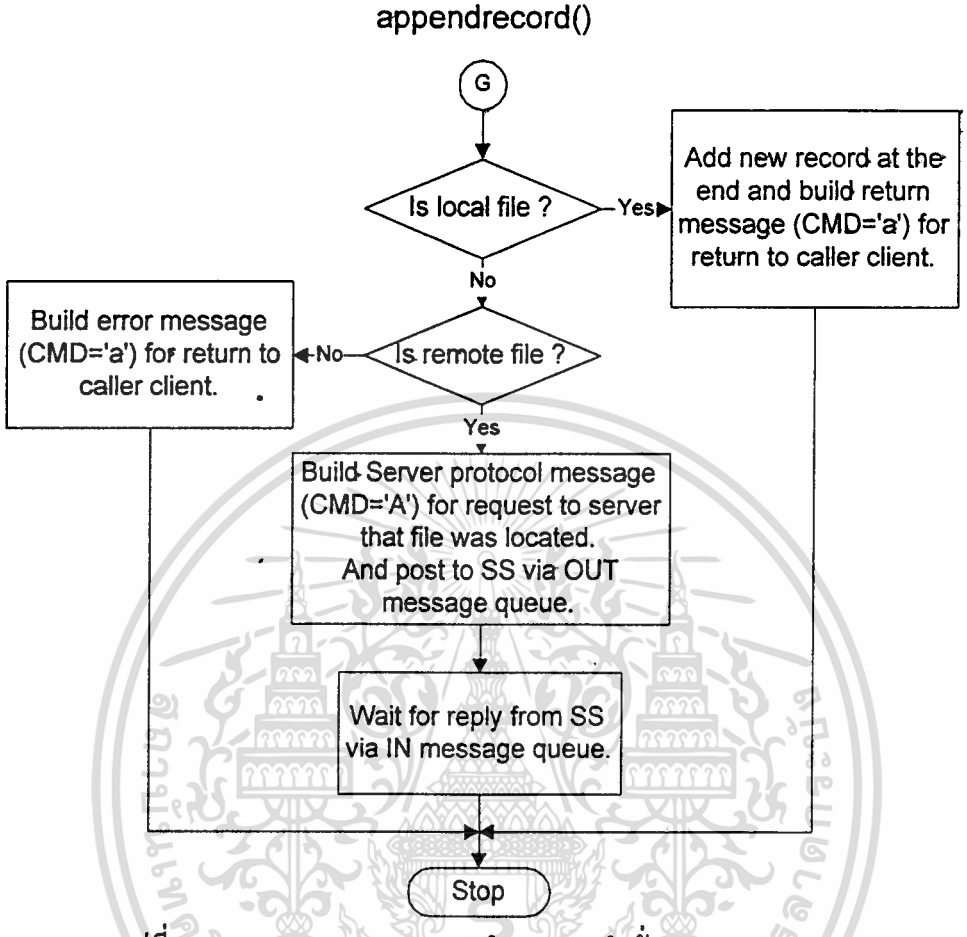
รูปที่ 3.1.10 flowchart แสดงการทำงานของคำสั่ง close table

List records จะต้องเช็คให้ได้ก่อนว่าไฟล์ที่ต้องการนั้นเป็น local หรือ remote ถ้าเป็น local ก็ทำการค้นหาข้อมูลนั้นได้เลยแล้วแจ้งผลให้ผู้ใช้ทราบผ่านโปรเซส client support แต่ถ้าไฟล์นั้นเป็น remote ก็จะต้องส่งเมสเซสนี้ไปให้เซอร์เวอร์อื่นๆที่มีไฟล์นั้นอยู่เป็นตัวจัดการ แล้วรอรับคำตอบจากโปรเซส server support แต่ถ้าไฟล์ที่ต้องการค้นหาข้อมูลนั้นไม่มีอยู่ทั้งใน local และ remote ก็ไม่สามารถทำงานได้



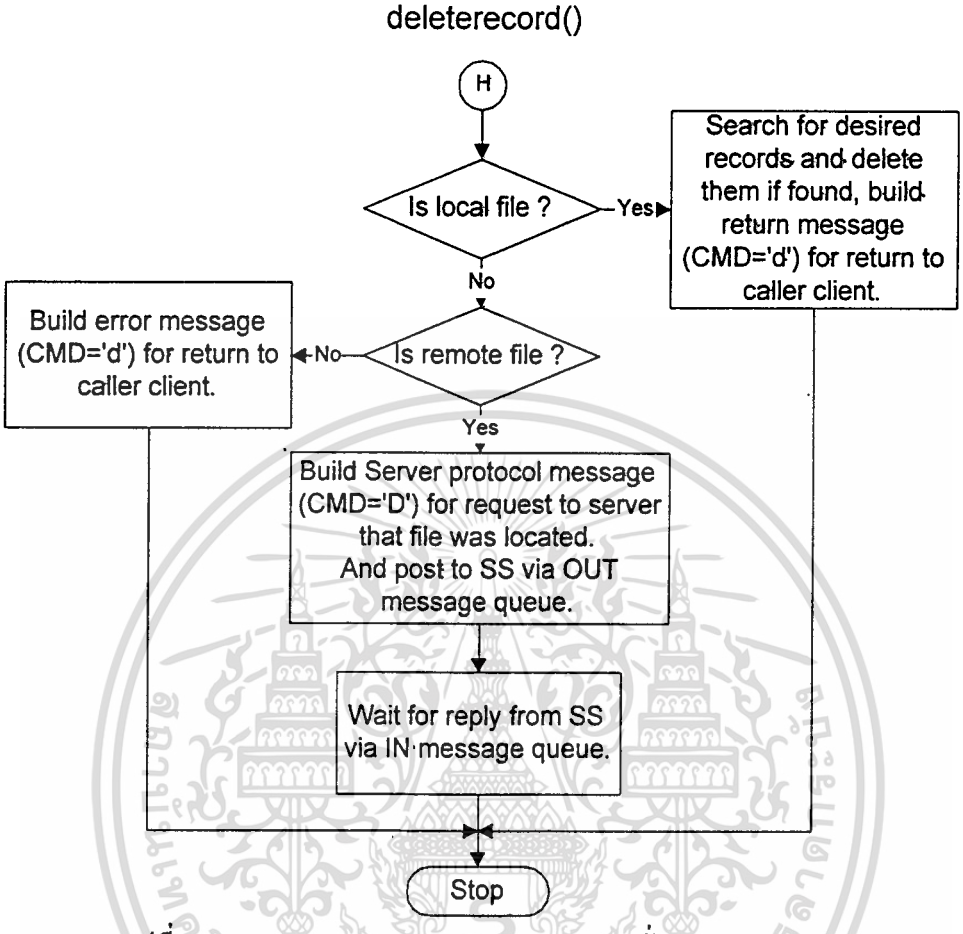
รูปที่ 3.1.11 flowchart แสดงการทำงานของคำสั่ง list records

Append record จะต้องเช็คให้ได้ก่อนว่าไฟล์ที่ต้องการนั้นเป็น local หรือ remote ถ้าเป็น local ก็ทำการเพิ่มเรคคอร์ดของข้อมูลนั้นได้เลยแล้วแจ้งผลให้ผู้ใช้ทราบผ่านโปรเซส client support แต่ถ้าไฟล์นั้นเป็น remote ก็จะต้องส่งเมสเซสนี้ไปให้เซอร์เวอร์อื่นๆที่มีไฟล์นั้นอยู่เป็นตัวจัดการแล้วรอรับคำตอบจากโปรเซส server support แต่ถ้าไฟล์ที่ต้องการค้นหาข้อมูลนั้นไม่มีอยู่ทั้งใน local และ remote ก็ไม่สามารถทำงานได้



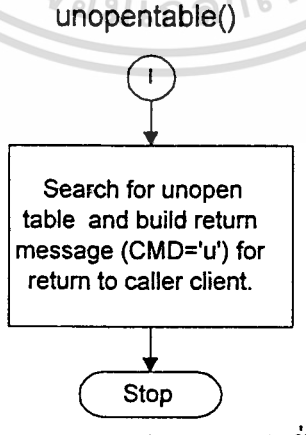
รูปที่ 3.1.12 flowchart แสดงการทำงานของคำสั่ง append records

Delete record จะต้องเช็คให้ได้ก่อนว่าไฟล์ที่ต้องการลบข้อมูลนั้นเป็น local หรือ remote ถ้าเป็น local ก็ทำการค้นหาและลบเรคคอร์ดของข้อมูลนั้นได้เลยแล้วแจ้งผลให้ผู้ใช้ทราบผ่านโปรเซส client support แต่ถ้าไฟล์นั้นเป็น remote ก็จะต้องส่งเมสเซสนี้ไปให้เซอร์เวอร์อื่นๆที่มีไฟล์นั้นอยู่เป็นตัวจัดการ แล้วรอรับคำตอบจากโปรเซส server support ผ่านทางเมสเซสคิว แต่ถ้าไฟล์ที่ต้องการลบข้อมูลนั้นไม่มีอยู่ทั้งใน local และ remote ก็ไม่สามารถทำงานได้



รูปที่ 3.1.13 flowchart แสดงการทำงานของคำสั่ง delete records

List unopen table จะต้องค้นหาไฟล์ที่ยังไม่ได้เปิดโดยเริ่มค้นหาจาก local ก่อน แล้วทำการส่งเมสเซสไปขอข้อมูลจากเซอร์เวอร์อื่นๆต่อไป



รูปที่ 3.1.14 flowchart แสดงการทำงานของคำสั่ง list unopen tables

List open table เป็นคำสั่งที่ไม่ต้องการติดต่อกับ file manager เพราะจะใช้ข้อมูลที่ถูกเก็บไว้ในขณะทำงานของตัวไคลแอนต์เอง

ในการติดต่อกันระหว่างผู้ให้บริการ (server) เพื่อแลกเปลี่ยนข้อมูลที่เรียกโดยผู้ใช้บริการ (client) นั้น จะกระทำโดย file manager ซึ่งจะเกิดขึ้นเมื่อมีการเรียกใช้ข้อมูลที่ไม่มีในเครื่องนั้นๆ จึงจำเป็นจะต้องเรียกจากผู้ให้บริการตัวอื่นโดยรูปแบบของการร้องขอจะเรียกว่า **server protocol**

3.1.7 Client protocol

client protocol จะใช้ในการติดต่อสื่อสารระหว่าง client process และ client support process ซึ่งจะเป็นรูปแบบของคำสั่งจากไคลแอนต์ ให้ file manager กระทำ โดยผ่าน client support process และในทางกลับกันที่ file manager จะตอบกลับไปที่ไคลแอนต์ค่าของข้อมูล (value) ในแต่ละข้อความ (message) จะเป็นตัวอักษร (Char) ทั้งหมด ทั้งนี้อาจจะเปลืองเนื้อที่บ้าง แต่เพื่อง่ายต่อการแสดงค่าของข้อมูล (Presentation layer) โดยมีรูปแบบของข้อความ (message format) โดยรวมดังนี้

Client ID	Command	Data

Client ID คือ หมายเลขประจำตัวของผู้ใช้บริการทุกตัว มีขนาด 3 ไบท์ เพราะฉะนั้นจึงรองรับผู้ให้บริการได้ 999 ตัว

Command คือ คำสั่งของข้อความนั้น เช่น เรียกดูข้อมูล (List) มีขนาด 1 ไบท์

Data คือ ข้อมูลเพิ่มเติมซึ่งจะมีขนาดต่างกันไปตามคำสั่ง อาจจะไม่มีก็ได้

รายละเอียดของแต่ละคำสั่งมีดังนี้

1. Create table

: เพื่อสร้างตารางฐานข้อมูลขึ้นใหม่ที่ Local server

- Request

CID	C	t_name	no_col	c_name1	c_type1	c_len1	.....	c_len*n*
3	1	8	1	8	1	2	.....	2

- Return

CID	c	t_name
3	1	8

Error : t_name = "00000000"

2. Delete table

: ลบตารางฐานข้อมูล

- Request

CID	E	t_name
3	1	8

- Return

CID	e	t_name
3	1	8

Error : t_name = "00000000"

3. Open table

: เพื่อเปิดฐานข้อมูลที่มีอยู่ขึ้นมาใช้งาน

- Request

CID	O	t_name
3	1	8

- Return

CID	o	tid	no_col	c_name1	c_type1	c_len1	.....	c_lenn
3	1	3	2	8	1	2	.....	2

Error : tid = 000

4. Close table

: เพื่อปิดตารางฐานข้อมูลที่เปิดแล้ว

- Request

CID	Q	tid
3	1	3

- Return

CID	q	tid
3	1	3

Error : tid = 0000

5. List record

: เพื่อแสดงข้อมูลในฐานข้อมูลที่เปิดแล้ว ตามเงื่อนไขที่ป้อน

- Request

CID	L	tid	col_val1	col_val2	col_val3	col_val4	.....	col_val*n*
3	1	3	x	x	x	x	.....	x

col_val*x* = * หมายถึงทุกค่า

- Return

CID	l	no_rec	col_val1	col_val2	col_val3	col_val4	.....	col_val*n*
3	1	4	x	x	x	x	.....	x

Last record : rec_no = 0000

6. Append record

: เพื่อเพิ่มข้อมูลเข้าไปในฐานข้อมูลที่เปิดแล้ว

- Request

CID	A	tid	col_val1	col_val2	col_val3	col_val4	.....	col_val*n*
3	1	3	x	x	x	x	.....	x

- Return

CID	a	tid
3	1	3

Error : tid = 000

7. Delete record

: เพื่อลบข้อมูลออกจากฐานข้อมูลที่เปิดแล้ว

- Request

CID	D	tid	col_val1	col_val2	col_val3	col_val4	.....	col_val*n*
3	1	3	x	x	x	x	.....	x

- Return

CID	d	no_rec
3	1	4

Error : no_rec = 0000

8. List unopen table

- Request

CID	U
3	1

- Return

CID	u	no_table	t_name1	t_name2	t_name3	t_name4	.....	t_namen
3	1	3	8	8	8	8	.....	8

Error : no_table = 000

9. Server support

- Request

CID	S	Server protocol
3	1	*

- Return

CID	s	Server protocol
3	1	*

10. Exit (Disconnect)

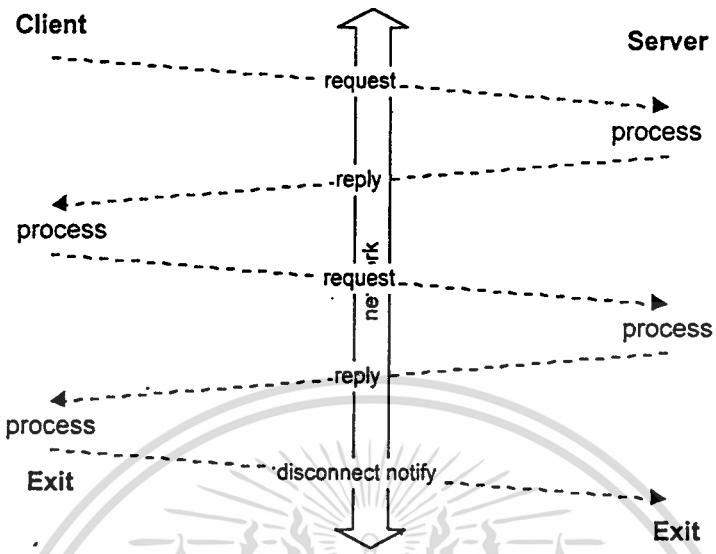
: เพื่อยกเลิกการติดต่อกับผู้ให้บริการ

- Request

CID	X
3	1

Return : None

การทำงานจะเริ่มเมื่อ client ติดต่อกับ client support process ได้แล้วฝ่ายผู้ให้บริการ(server) ก็จะรอการร้องขอจากผู้ใช้บริการ (client) และตอบรับตามข้อความที่ร้องขอเข้ามา จนกระทั่งผู้ใช้บริการบอกเลิกการติดต่อ การติดต่อจึงสิ้นสุดลง พอจะแสดงเป็นแผนภาพได้ดังนี้



รูปที่ 3.1.15 ลักษณะการติดต่อสื่อสารระหว่างผู้ให้และผู้ใช้บริการ

3.1.8 Server protocol

server protocol เป็นรูปแบบและขั้นตอนการติดต่อกันระหว่างผู้ให้บริการด้วยกันเองเพื่อแลกเปลี่ยนข้อมูลที่ไม่มีในเครื่องตัวเอง โดยจะเป็นการติดต่อแบบ Connectionless ส่วนที่ทำงานนี้คือ Server support process รูปแบบของข้อความที่ส่งผ่านถึงกันเพื่อร้องขอใช้บริการและตอบรับนั้นมีลักษณะทั่วไปดังนี้

Source machine	Destination machine	CID	Command	Data

จะเห็นว่ามีลักษณะคล้ายกับ client protocol เพียงแต่เพิ่มช่องชื่อของผู้รับและผู้ส่งเท่านั้น คำสั่งทั้งหมด มีดังนี้

1. Delete table

: ลบแฟ้มข้อมูลที่มีอยู่

- Request

Source_IP	Dest_IP	CID	E	t_name
15	15	3	1	8

- Return

Source_IP	Dest_IP	CID	e	t_name
15	15	3	1	8

error : t_name = "00000000"

2. Open table

: เปิดตารางฐานข้อมูลเพื่อจัดการกับข้อมูล

- Request

Source_IP	Dest_IP	CID	O	t_name
15	15	3	1	8

- Return

Source	Dest.	CID	o	tid	no_col	c_name1	c_type1	c_len1	.....	c_lenn
15	15	3	1	3	2	8	1	2	.....	2

Error : tid = 000

3. Close table

: เพื่อปิดตารางฐานข้อมูลที่เปิดแล้ว

- Request

Source_IP	Dest_IP	CID	Q	tid
15	15	3	1	3

- Return

Source_IP	Dest_IP	CID	q	tid
15	15	3	1	3

Error : tid = 000

4. List record

: เพื่อแสดงข้อมูลในฐานข้อมูลที่เปิดแล้ว ตามเงื่อนไขที่ป้อน

- Request

Source	Dest	CID	L	tid	col_val1	col_val2	col_val3	.....	col_val n
15	15	3	1	3	x	x	x	.....	x

col_valx = * หมายถึงทุกค่า

- Return

Source	Dest	CID	l	rec_no	col_val1	col_val2	col_val3	.....	col_valn
15	15	3	1	4	x	x	x	.....	x

Last record : rec_no = 0000

5. Append record

: เพื่อเพิ่มข้อมูลเข้าไปในฐานข้อมูลที่เปิดแล้ว

- Request

Source	Dest	CID	A	tid	col_val1	col_val2	col_val3	.....	col_valn
15	15	3	1	3	x	x	x	.....	x

- Return

Source	Dest	CID	a	tid
15	15	3	1	3

Error : tid = 000

6. Delete record

: เพื่อลบข้อมูลออกจากฐานข้อมูลที่เปิดแล้ว

- Request

Source	Dest	CID	D	tid	col_val1	col_val2	col_val3	.....	col_val n
15	15	3	1	3	x	x	x	.....	x

- Return

Source	Dest	CID	d	rec_no
15	15	3	1	4

Error : rec_no = 0000

7. Broadcast request

: เป็นการกระจายข้อมูลจากผู้ให้บริการหนึ่งไปสู่ผู้ให้บริการตัวอื่นๆ เพื่อสื่อสารข้อมูลกัน

Source	Dest	CID	B	no_table	t_name1	tid1	no_col1	col_name11	col_type11
15	15	3	1	2	8	3	2	8	1

col_len11	col_name12	col_type12	col_len12	.....	t_name2	tid2	no_col2
2	8	1	2	.....	8	3	2

col_name21	col_type21	col_len21	.....
8	1	2	.....

8. Add new table

: เป็นการส่งข้อมูลจากผู้ให้บริการตัวหนึ่งไปยังผู้ให้บริการตัวอื่น เมื่อมีการสร้างแฟ้มข้อมูลขึ้นใหม่

Source	Dest	CID	N	no_table	t_name1	tid1	no_col1	col_name11	col_type11
15	15	3	1	2	8	3	2	8	1

col_len11	col_name12	col_type12	col_len12	....	t_name2	tid2	no_col2
2	8	1	2	....	8	3	2

col_name21	col_type21	col_len21	.....
8	1	2	.....

9. Delete table from structure

: เมื่อมีการลบแฟ้มข้อมูลออก ผู้ให้บริการซึ่งมีแฟ้มข้อมูลนั้นอยู่จะต้องบอกให้ผู้บริการตัวอื่นๆทราบ เพื่อทำการ update ข้อมูล

Source	Dest	CID	F	no_table	t_name1	tid1	no_col1	col_name11	col_type11
15	15	3	1	2	8	3	2	8	1

col_len11	col_name12	col_type12	col_len12	.....	t_name2	tid2	no_col2
2	8	1	2	.....	8	3	2

col_name21	col_type21	col_len21	.....
8	1	2	.....

บทที่ 4

ขั้นตอนการดำเนินงาน

1. ทำการศึกษาทฤษฎีต่างๆที่เกี่ยวข้องและแนวทางที่เป็นไปได้ในการจัดทำ โดยได้ศึกษาวิธีการเขียนโปรแกรมทั้งสองลักษณะคือ ติดต่อผ่าน RPC และการติดต่อกับชั้นที่ 4 ของ OSI Model โดยตรง โดยใช้ BSD Socket และ System V TLI

2. ตัดสินใจเลือกวิธีที่จะใช้พัฒนาโครงงาน โดยเลือกวิธีการเขียนโปรแกรมแบบใช้ชุดคำสั่งของ BSD Socket และ System V TLI ในการตัดสินใจเลือกการติดต่อกับชั้นที่ 4 ของ OSI Model โดยตรงนั้น เพราะว่าการเขียนโปรแกรมในลักษณะนี้มีความยืดหยุ่นมากกว่าและมีข้อจำกัดที่น้อยกว่าการเขียนผ่าน RPC กล่าวคือ RPC มีเฉพาะระบบปฏิบัติการ UNIX เท่านั้น ไม่สามารถเขียนโปรแกรมไปติดต่อกับระบบอื่นได้ เช่น ดอส วินโดว์ เป็นต้น

3. ออกแบบโครงสร้างของระบบ (System architecture) หน้าที่ของแต่ละส่วน (Modules function) และข้อตกลงที่จะใช้ติดต่อกันระหว่างส่วนต่างๆในระบบ (Protocol) ซึ่งมีสองแบบ คือ ระหว่างตัวจัดการแฟ้มข้อมูล (file manager) ด้วยกันเองเรียกว่า Server protocol และระหว่างผู้ใช้(client) กับผู้ให้บริการ(server) เรียกว่า Client protocol

4. ทดลองสร้างโปรแกรมเพื่อรับส่งข้อมูลแบบง่ายๆระหว่างยูนิกส์กับยูนิกส์และยูนิกส์กับไมโครซอร์ฟวินโดว์ และโปรแกรมที่มีการติดต่อกันเองภายในเครื่องเดียวกันโดยใช้ ไปป์และเมสเสสคิว

5. นำผลของการทดลองมาปรับปรุงเปลี่ยนแปลงโครงสร้างของระบบ โดยโปรแกรมที่มีการติดต่อกันเองภายในเครื่องเดียวกันจะใช้เมสเสสคิว เนื่องจากมีประสิทธิภาพและอำนวยความสะดวกในการเขียนโปรแกรมมากกว่าไปป์

6. เขียนโปรแกรมเพื่อสร้างระบบตามแบบที่ได้กำหนดไว้โดยเริ่มจากบนระบบปฏิบัติการยูนิกซ์ (BSD socket)

7. ทำการแก้ไขปรับปรุงการทำงานในส่วนต่างๆจากที่เคยออกแบบไว้ เช่น Server protocol และ Client protocol โดยเพิ่มรูปแบบของคำสั่งในการติดต่อให้มีมากขึ้น

8. ทำการเพิ่มความสามารถต่างๆของระบบให้ดียิ่งขึ้น เช่น การตรวจเช็คข้อผิดพลาด การติดต่อกับผู้ใช้งาน

9. ขยายความสามารถของระบบโดยเขียนโปรแกรมให้สามารถติดต่อใช้แฟ้มข้อมูลได้จากระบบปฏิบัติการอื่นๆ เช่น ไมโครซอฟวินโดว์ ยูนิกซ์ System V (TLI)

บทที่ 5

วิจารณ์และสรุปผลการทำงาน

4.1 สรุปผลการทำงาน

โครงงานนี้มีผลการดำเนินงานสำเร็จตามวัตถุประสงค์และจุดมุ่งหมายที่ไว้วางไว้ตั้งแต่แรก กล่าวคือได้ระบบจัดการแฟ้มข้อมูลอย่างง่ายที่สามารถทำงานได้ทั้งบนระบบปฏิบัติการยูนิกซ์ (BSD socket) และ (system V TLI) และบนไมโครซอฟวินโดว์

4.2 แนวทางการพัฒนาต่อ

โครงงานนี้ยังไม่สามารถนำไปใช้งานในปัจจุบันนี้ได้ดีนัก เพราะว่ายังขาดคุณสมบัติอีกหลายๆประการของระบบฐานข้อมูลแบบกระจาย เช่น กฎความคงสภาพของฐานข้อมูล การจัดการข้อมูลเมื่อมีผู้ใช้เข้าใช้แฟ้มข้อมูลเดียวกันในเวลาเดียวกันหลายคน ตลอดจนโปรแกรมช่วยเหลือต่างๆที่จัดการกับข้อมูล จึงควรเพิ่มประสิทธิภาพในส่วนที่กล่าวมา นอกจากนี้แล้วควรขยายขีดความสามารถของระบบขึ้นไปอีก เช่น สามารถใช้งานได้บนระบบปฏิบัติการ DOS และ OS/2

ภาคผนวก

สำหรับ source code listing และหนังสือคู่มือการใช้งานของโครงงานนี้ สามารถขอทำสำเนาได้จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

กิตติกรรมประกาศ

ทางคณะผู้จัดทำขอขอบพระคุณบุคคลต่างที่ได้ให้ความช่วยเหลือในการทำโครงงานนี้ให้สำเร็จบรรลุผลไปด้วยดี

1. บิดา มารดา ที่ให้ความสนับสนุนในการศึกษามาตลอด

2. อาจารย์ ดร.บุญธีร์ เครือตราชู ที่ช่วยให้คำปรึกษาแนะนำและข้อมูลที่มีประโยชน์ต่อโครงงาน ตลอดจนให้ยืมอุปกรณ์และตำรับตำราต่างๆ

3. คณะอาจารย์ที่ได้เคยถ่ายทอดความรู้ให้กับคณะผู้จัดทำ

4. เพื่อนๆที่ให้คำแนะนำติชมและให้กำลังใจ

5. บริษัท ยิบอินซอย จำกัด ที่ให้ยืมอูปกรณ์เครือข่าย ตลอดจนหนังสือต่างๆ

6. คุณตี๋กลาง ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ ที่ให้ยืมอุปกรณ์เครือข่าย

7. บุคคลต่างๆที่ได้ถ่ายทอดความรู้ต่างๆในการเขียนโปรแกรมลงบนอินเตอร์เน็ต

8. คณะบุคคลต่างๆที่ได้คิดชุดคำสั่งที่ใช้ในการเขียนโปรแกรมติดต่อกับเครือข่ายทุกท่าน

หนังสืออ้างอิง

1. W. Richard Stevents 1991, "UNIX Network Programming", Prentice-Hall International, Inc., New Jersey, 1991.

2. Ralph Davis 1993, "Windows Network Programming", Addison-Wesley 1993.

3. Andrew S. Tanenbaum 1992, "Modern Operating Systems", Prentice-Hall, International, Inc., New Jersey, 1992.

4. Douglas E. Comer 1991, "Internetworking With TCP/IP Vol I", Prentice-Hall International, Inc., New Jersey, 1991.

5. Douglas E. Comer, David L. Stevents 1991, "Internetworking With TCP/IP Vol II", Prentice-Hall International, Inc., New Jersey, 1991.

6. Douglas E. Comer, David L. Stevents 1993,"Internetworking With TCP/IP Vol III", Prentice-Hall International, Inc., New Jersey, 1993.