ระบบประมวลผลแบบกระจายสอดคล้อง โดยใช้ RPC

Synchronous Distributed Processing using RPC

โดย

นาย พรเทพ นฤหล้า

นาย ภูวดล ไชยภูริพัฒน์

ได้รับทุนอุดหนุนโครงงานซอฟท์แวร์ขนาดเล็กจาก

ศูนย์เทคโนโลยีอีเล็คทรอนิกส์และคอมพิวเตอร์แห่งชาติ

สำนักพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2536

ปริญญานิพนธ์ปีการศึกษา 2536

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบประมวลผลแบบกระจายสอดคล้องโดยใช้ RPC

ผู้จัดทำ

1. นาย พรเทพ นฤหล้า 33100239
2. นาย ภูวดล ไชยภูริพัฒน์ 33100290

_____ อาจารย์ที่ปรึกษา

(ดร. บุญธีร์ เครือตราชู)

_____ อาจารย์ที่ปรึกษา

(อ. อภิเนตร อูนากูล)

ระบบประมวลผลแบบกระจายสอดคล้องโดยใช้ RPC

Synchronous Distributed Processing using RPC

โดย นาย พรเทพ นฤหล้า

นาย ภูวดล ไชยภูริพัฒน์

อาจารย์ที่ปรึกษา ดร. บุญธีร์ เครือตราชู

อ. อภิเนตร อูนากูล

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของโครงงานวิจัยเพื่อทำการศึกษาระบบประมวลผลแบบกระจาย (Distributed Processing) โดยทำการศึกษาปัจจัยที่ต้องคำนึงถึงในการพัฒนาระบบประมวลผลแบบกระจายสองปัจจัย คือ การควบคุมความสอดคล้อง (Synchronization Control) และ การควบคุมความถูกต้องตรงกันของข้อมูล (Consistency Control) โดยได้ใช้การพัฒนาโปรแกรมเกมแบบผู้เล่นหลายคนบนเครือข่ายซึ่งมีระบบการทำงานแบบเรียลไทม์ (real-time) เป็นกรณีศึกษา ซึ่งได้ใช้หลักการ การใช้หน่วยความจำร่วมกัน (shared-memory) การส่งผ่านข้อมูลและคำสั่ง (message passing protocol) ด้วยรีโมทโพรซีเจอร์คอลลิง (RPC) และ การตรวจสอบสถานะของข้อมูลโดยใช้แฟลกบอกการเปลี่ยนแปลง และการบันทึกเวลา

ABSTRACT

This thesis deals with the study of Synchronous Distributed Processing system by using a multi-player game running on a network, a real-time application, as the case-study.

The project focuses onto two of the concerned issues in developing a distributed processing system which are Synchronization Control and Consistency Control, using several multi-programming methods such as shared-memory, remote procedure calling (RPC), changed-flag variable, and time accounting to achieve the solutions for both issues.

สารบัญ

สารบัญรูปภาพ ตาราง และ ตัวอย่าง

บทที่ 1
บทนำ

ปัจจุบันเทคโนโลยีของระบบการประมวลผลแบบกระจาย (Distributed Processing) ได้พัฒนาก้าวหน้าไปอย่างมาก และได้ถูกนำไปใช้กับงานหลายๆ ประเภท โดยส่วนมากจะเป็นงานที่ต้องมีการประมวลผลข้อมูลปริมาณมหาศาล เช่น การทำ Image Processing เป็นต้น ซึ่งจะช่วยย่นระยะเวลาในการคำนวณให้สั้นลงอย่างมาก หรืองานพไม่ต้องใช้การคำนวณสูงแต่ต้องมีการคำนวณและควบคุมอุปกรณ์หลายๆ อย่างเช่น การควบคุมเครื่องจักรหลายๆ ตัวในโรงงานที่ต้องทำงานสอดคล้องกัน ก็ถือว่าเป็นลักษณะของการประมวลผลแบบกระจายเช่นเดียวกัน

ในการพัฒนาระบบการประมวลผลแบบกระจายนั้นมีปัจจัย (Concerned issues) สำคัญที่จะต้องคำนึงถึงอยู่ดังนี้

- ☐ Task Decompositon : การแบ่งงานออกเป็นส่วนย่อยๆเพื่อนำไปจ่ายให้หน่วยประมวลผลที่มีอยู่เพื่อให้ได้ประสิทธิภาพการทำงาน (efficiency) และ ใช้ประโยชน์ของทรัพยากรอย่างคุ้มค่า (resource utilization)
- ☐ Synchronization Control : การควบคุมความสอดคล้องในการประมวลผลระหว่างแต่ละโปรเซส (Process) ที่ทำงานแต่ละส่วนซึ่งอาจจะมีการถ่ายเทข้อมูลซึ่งกันและกัน
- ☐ Data Consistancy Control : คือการควบคุมความถูกต้องของข้อมูลที่ใช้ในการประมวลผล โดยข้อมูลหลังจากที่ทำการประมวลผลแบบกระจายแล้วนั้นจะต้องมีค่าเหมือนกับข้อมูลที่ประมวลผลบนเครื่องเครื่องเดียว
- ☐ Security/Authentication Control : ความปลอดภัยของข้อมูลที่ส่งไปประมวลผลยังเครื่องอื่นๆ

1.1 วัตถุประสงค์

โครงงานนี้เป็นการศึกษาเพื่อออกแบบและทดสอบวิธีการควบคุมความสอดคล้องในการประมวลผลและการควบคุมความถูกต้องตรงกันของข้อมูลขณะที่ทำการประมวลผลแบบกระจาย โดยใช้เกมแบบผู้เล่นหลายคนบนเครือข่ายเป็นกรณีศึกษา (Case Study) ซึ่งเป็นตัว

อย่างที่ทำให้เห็นผลการทำงานของเทคนิคที่ใช้ได้อย่างชัดเจน นอกจากนั้นยังเป็นการศึกษาวิธีการรับส่งข้อมูลผ่านเครือข่าย และ การศึกษาการพัฒนาโปรแกรมบน X Window อีกด้วย

1.2 ขั้นตอนการทำงาน

เริ่มจากการศึกษาหาข้อมูลเกี่ยวกับเรื่องการประมวลผลแบบกระจาย และได้เลือกศึกษาวิธีควบคุมความสอดคล้องในการประมวลผล และ ารควบคุมความถูกต้องตรงกันของข้อมูล หลังจากนั้นได้ทำการหาลักษณะของโปรแกรมที่เหมาะสมที่จะใช้เป็นกรณีศึกษา จึงได้เลือกเกมระบบ Real Time ขั้นต่อมาเป็นการวางแผนการทำงานและออกแบบโครงสร้างของโปรแกรม แล้วทำการแบ่งการทำงานออกเป็นสองช่วง โดยช่วงแรกเป็นการศึกษาเพื่อออกแบบและสร้างฟังก์ชันพื้นฐานที่ใช้ในการติดต่อสื่อสารผ่านเครือข่าย ซึ่งจะต้องมีการควบคุมความถูกต้องและความเร็วในการรับส่งข้อมูล ส่วนในช่วงที่สองก็ยังได้แบ่งการทำงานออกเป็นสองส่วนคือ การศึกษาและสร้างฟังก์ชันในส่วนการติดต่อกับผู้เล่น(User Interface) โดยใช้ X Window และการศึกษาและออกแบบวิธีการควบคุมความสอดคล้องในการประมวลผล และความถูกต้องของข้อมูล และส่วนปลีกย่อยอื่นๆ

บทที่ 2
ทฤษฎีและหลักการ

2.1 การประมวลผลแบบกระจาย

การประมวลผลแบบกระจาย เป็นสาขาวิชาย่อยที่เป็นส่วนหนึ่งของการศึกษาระบบการประมวลผลแบบขนาน (Parallel Processing) ซึ่งเป็นการจัดแบ่งงาน (task) หนึ่งงาน หรือหลายงานให้หน่วยประมวลผล (Processors) หลายๆหน่วยร่วมกันประมวลผลในลักษณะขนาน ซึ่งแตกต่างจากการนำหน่วยประมวลผลเพียงหน่วยเดียวมาจัดแบ่งเวลา (Schedule) ให้ทำการประมวลผลหลายๆงานในลักษณะ Time-sharing ที่เรียกว่าการทำ multi-tasking

สำหรับสถาปัตยกรรมของระบบการประมวลผลแบบขนาน (Parallel Architecture) นั้นมีที่ใช้อยู่สองแบบได้แก่

- ☐ SIMD (Single instruction stream, multiple data stream) เป็นการนำข้อมูลหลายชุดมาประมวลผลพร้อมกันด้วยขั้นตอนการทำงานเดียวกัน เช่น ระบบ vector processing เป็นต้น
- ☐ MIMD (Multiple instruction stream, multiple data stream) เป็นการแยกงานชนิดต่างๆ กันที่ทำการประมวลผลกับข้อมูลต่างชนิดกันไปทำในหน่วยประมวลผลหลายหน่วย

ปัจจัยที่ต้องคำนึงถึงในการออกแบบระบบประมวลผลแบบกระจาย (Concerned issues)

- ☐ Task Decomposition คือ รูปแบบ/วิธีการที่ใช้ในการแบ่งแยกงานให้แต่ละหน่วยประมวลผลนั่นเอง ซึ่งมีอยู่สามรูปแบบดังนี้
  1. Completely Parallel
  2. Domain Decompostion
  3. Control Decomposition
- ☐ Synchronization คือ วิธีที่ใช้ในการควบคุมการทำงานของหน่วยประมวลผลให้สอดคล้องกัน (เข้าจังหวะกัน)

- ☐ Data consistency คือ การควบคุมให้ผลการประมวลผลที่ได้นั้นถูกต้องเหมือนกับเมื่อใช้การประมวลผลแบบตามลำดับ
- ☐ Security/Authentication คือ การป้องกันระบบจากผู้ที่ไม่ได้รับอนุญาตให้ใช้ระบบ และ การตรวจสอบได้ว่าผู้ใช้ที่เข้ามาใช้ระบบนั้นเป็นผู้ใช้ที่ได้รับอนุญาตจริง (user authenticable)

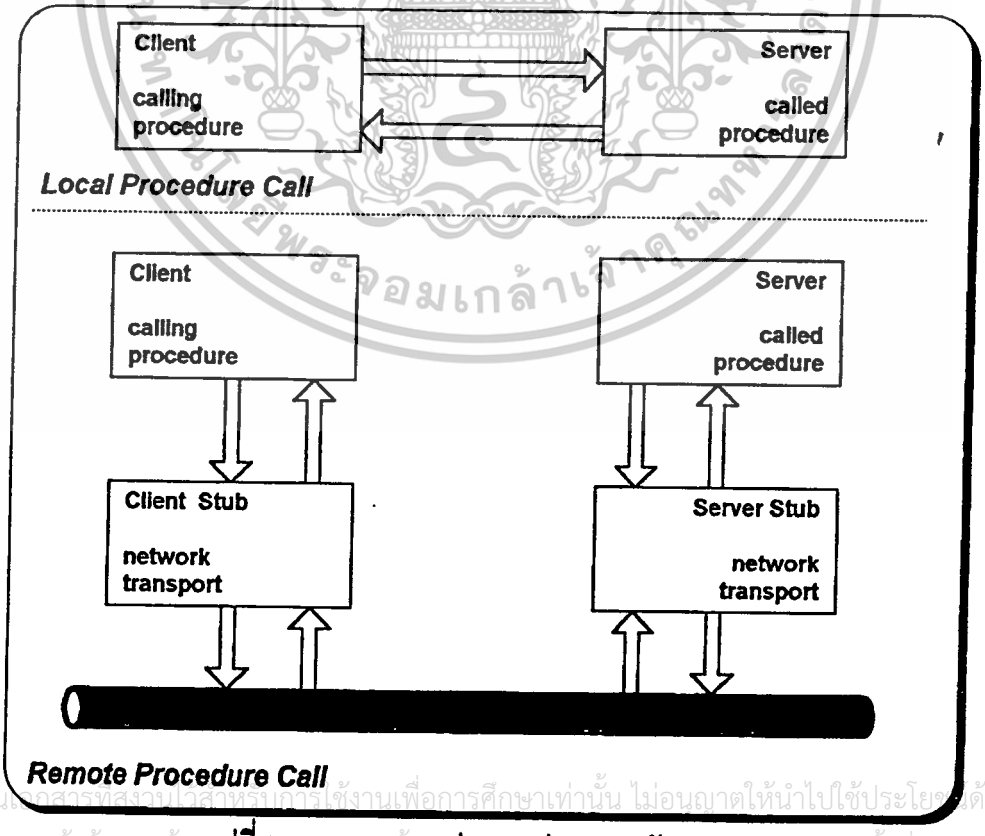
สำหรับทฤษฎีการพัฒนาระบบประมวลผลแบบกระจายโดยละเอียด และตัวอย่างการนำไปใช้งานนั้นสามารถศึกษาเพิ่มเติมได้จากหนังสืออ้างอิง [11] ที่ได้ระบุไว้ท้ายเล่มต่อไป

2.2 Remote Procedure Calls (RPC)

การใช้ RPC ทำให้สามารถสร้างโปรแกรมให้ ผู้เรียก (Client) สามารถเรียกใช้ procedure ที่อยู่บนเครื่องคอมพิวเตอร์เครื่องอื่นบนเครือข่าย ด้ โดยที่ RPC นั้น นอกจากจะเป็นพื้นฐานของระบบการประมวลผลแบบกระจาย (Distributed processing System) ที่ใช้อยู่ในปัจจุบันเช่น NFS และ NIS แล้ว ผู้ใช้ (programmer) ยังสามารถนำ RPC ไปใช้ในการพัฒนาโปรแกรมแบบ client/server ได้อีกด้วย โดยที่ RPC นั้นจะช่วยให้การพัฒนาโปรแกรมแบบ client/server มีประสิทธิภาพสูงขึ้น และ สะดวกสบายขึ้นกว่าการใช้ Socket Interface และเมื่อใช้ร่วมกับ ONC RPCGEN โปรโตคอลคอมไพเลอร์ (Protocol Compiler) แล้วก็จะทำให้การเรียก remote procedure มีลักษณะเหมือนกับการเรียก local procedure ธรรมดาๆ

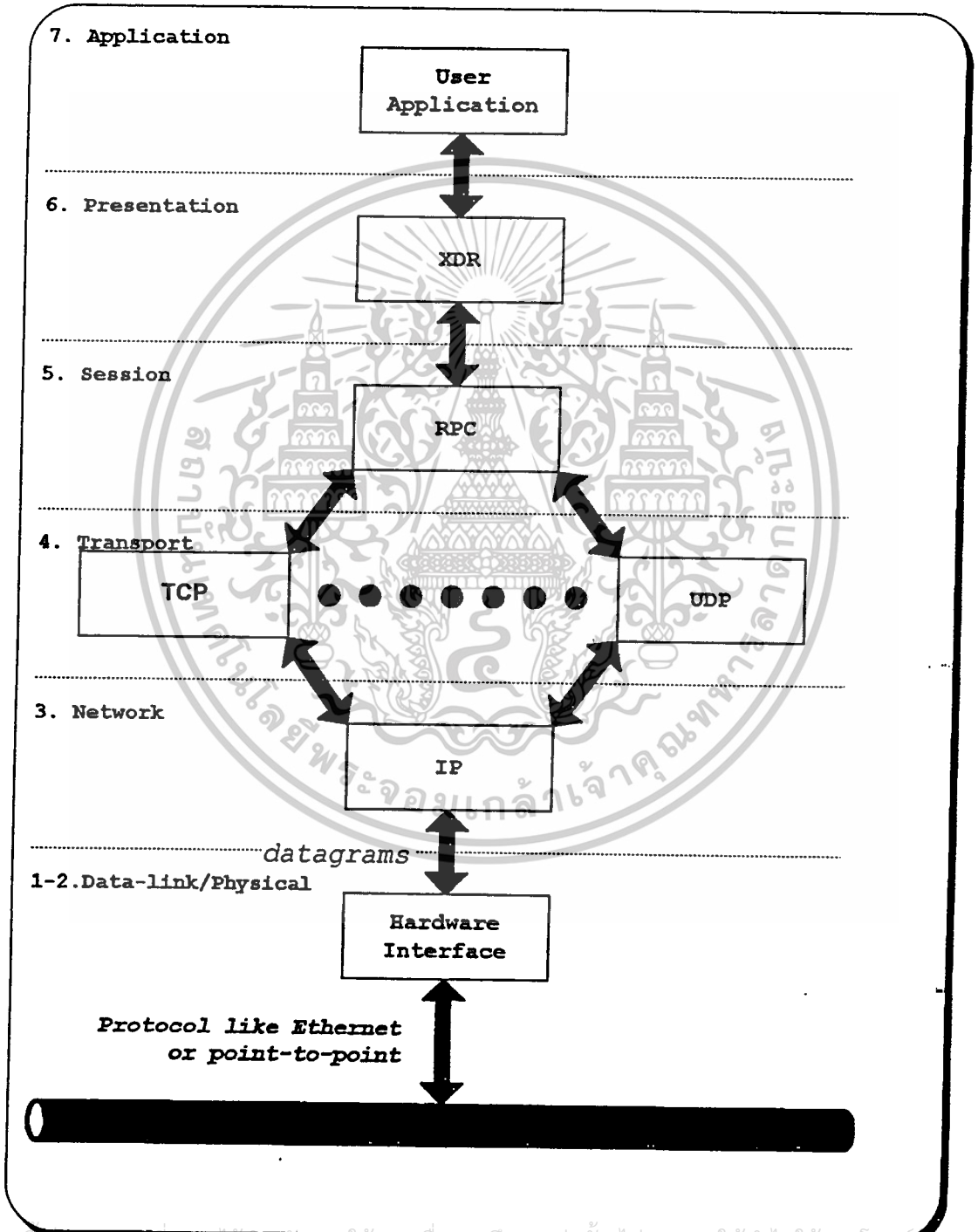
2.2.1) ความแตกต่างระหว่าง local procedure calling และ remote procedure calling

จากรูปที่ 2.2-1 จะเห็นได้ว่าในการใช้ RPC นั้น ผู้เรียก (Client) และ ผู้ให้บริการ (Server) นั้นจะวิ่ง (run) อยู่บนคนละโปรเซส และ ยังอาจอยู่บนคนละเครื่องบนเนตเวอร์คอีกด้วย ซึ่ง RPC libraries จะรับหน้าที่ในการติดต่อระหว่างโปรเซสผู้เรียก และ โปรเซสผู้ให้บริการ และการแปลงรูปแบบข้อมูล (data type conversion) ที่จะต้องส่งผ่านเครือข่าย เช่น parameter



รูปที่ 2.2-1 ความแตกต่างระหว่าง LPC กับ RPC

และ return argument เป็นต้น ให้อยู่ในรูปที่เรียกว่า External Data Representation(XDR) โดยที่การทำงานของ XDR และ RPC นั้น จะอยู่ใน Presentation Layer และ Session Layer ของ OSI ตามลำดับดังแสดงไว้ในรูปที่ 2.2-2



รูปที่ 2.2-2 ตำแหน่งของ RPC และ XDR ใน OSI Model

ส่วนของโปรแกรมที่ทำหน้าที่จัดการการติดต่อระหว่าง โปรเซสผู้เรียก และโปรเซสผู้ให้บริการ และทำการแปลงข้อมูลนั้นเรียกว่า stub ของผู้เรียก และ stub ของผู้ให้บริการตามลำดับ โดย RPC โปรโตคอลคอมไพเลอร์ เช่น ONC RPCGEN จะทำหน้าที่ในการสร้าง code ส่วน stub นี้ขึ้นมาเพื่อนำไป link กับ code ส่วนของผู้เรียก และ ผู้ให้บริการอีกทีหนึ่ง

2.2.2) การพัฒนา Application ที่ใช้ RPC

การพัฒนาโปรแกรมที่ใช้ RPC จะประกอบด้วยขั้นตอนดังต่อไปนี้ คือ

- ☐ กำหนด โปรโตคอล ที่จะใช้ในการติดต่อระหว่าง ผู้เรียก และ ผู้ให้บริการ
- ☐ เขียน code ส่วนของ ผู้เรียก และ ผู้ให้บริการ
- ☐ จากนั้นจะเป็นการ compile แล้ว link เข้ากับ stub และ libraries แล้วจึงทำการทดสอบโดยการ run โปรเซสผู้ให้บริการบนเครื่องๆหนึ่ง และ run โปรเซส ผู้เรียกบนอีกเครื่องหนึ่งบนเครือข่าย เพื่อตรวจสอบผลการทำงาน

2.2.2.1) การกำหนด RPC โปรโตคอล

ขั้นแรกของการสร้างโปรแกรมด้วย RPC จะต้องทำการกำหนดชื่อของ procedures และ ชนิดของ parameter และ return argument ที่จะอนุญาตให้ผู้เรียกสามารถเรียกใช้ได้ ซึ่ง โปรโตคอลคอมไพเลอร์ เช่น ONC RPCGEN เป็นต้น จะใช้ข้อกำหนดโปรโตคอล (Protocol Definitions) ที่เรากำหนดไว้นี้ในการสร้าง stub ทั้งส่วนของผู้เรียก และ ส่วนของผู้ให้บริการ รวมทั้งส่วน XDR filter ออกมาให้

ภาษาที่ใช้ในการกำหนดโปรโตคอล ของ RPC นั้นเรียกว่า ONC RPC Language(RPCL) ซึ่งมีไวยากรณ์บางส่วนคล้ายกับภาษา C และ Pascal โดยที่ประกอบไปด้วยข้อกำหนดต่างๆ ซึ่งได้แก่

- ☐ Constant *(const)*
- ☐ Enumerations *(enum)*
- ☐ Structure *(struct)*
- ☐ Union *(union)*
- ☐ Typedefinition *(typedef)*

☐ Program (*program*)

enum, struct และ *typedef* นั้นจะเหมือนกับที่ใช้ในภาษ C และ *const* นั้นใช้ในการกำหนดชื่อ (label) ให้กับค่าคงที่แบบจำนวนเต็ม ส่วน *union* ของ RPCL นั้นจะเหมือนกับ variant record ในภาษา Pascal และ *program* จะเป็นส่วนที่ใช้ในการประกาศ procedure ต่างๆที่

ตัวอย่างที่ 2.2-1: ตัวอย่างข้อกำหนดโปรโตคอล RPCL: rdb.x

```
/* preprocessor directives */
%#define DATABASE "personnel.dat"

/* constant definitions */
const MAX_STR      = 256;

/* structure definitions, no enumerations needed */
struct record {
      string      firstName<MAX_STR>;      /* <> defines
                        the maximum possible length*/
      string      moddleInitial<MAX_STR>;
      string      lastName<MAX_STR>;
      int   phone;
      string      location<MAX_STR>;
};

/* program definition, no union or typedef definitions
      needed */
program RDBPROG { /* could manage multiple servers */
      version RDBVERS {
            record FIRSTNAME_KEY(string)   = 1;
            record LASTNAME_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 established by
                        ONC */
```

จะอนุญาตให้เรียกใช้ได้ ซึ่งไฟล์ที่ใช้เก็บข้อกำหนดโปรโตคอล นี้ จะมีนามสกุลต่อท้ายด้วย ".x" ซึ่งตัวอย่างของข้อกำหนดโปรโตคอล ได้แสดงไว้ใน ตัวอย่างที่ 2.2-1 ซึ่งจะเห็นได้ว่ามี procedure ที่ใช้ในการค้นหา และ เพิ่ม record ใน ฐานข้อมูลอยู่ห้า procedure ด้วยกัน โดยแต่ละ procedure จะมีหมายเลขประจำตัวซึ่งเริ่มจาก 1 เป็นต้นไป ส่วน procedure หมายเลข 0 จะถูกเพิ่มเข้าไปโดยอัตโนมัติในระหว่างการcompile โดย RPCGEN เพื่อให้ใช้ในการทดสอบ (ping) ได้ว่า ขณะนั้นโปรเซสผู้ให้บริการยังวิ่งอยู่หรือไม่

การ compile ข้อกำหนดโปรโตคอล จะใช้คำสั่ง rpcgen ดังตัวอย่าง:

$ *rpcgen rdb.x*

ซึ่งจะได้ไฟล์ stub code ของผู้เรียก ชื่อ rdb_clnt.c ไฟล์ stub code ของผู้ให้บริการ ชื่อ rdb_svc.c ไฟล์ XDR filter code ชื่อ rdb_xdr.c และไฟล์ header ที่ทั้งส่วนผู้เรียก และผู้ให้บริการจะต้อง #include ชื่อ rdb.h ซึ่งชื่อ procedure จะถูกแปลงให้เป็นอักษรตัวเล็ก และ ต่อท้ายด้วย '_' และ หมายเลข version ซึ่งในที่นี้เท่ากับ 1 ส่วน หมายเลขประจำตัวของแต่ละ procedure ที่จะใช้ในการเรียกได้ถูกกำหนดในรูปแบบ long integer

ส่วน stub ของผู้เรียกที่ได้จาก rpcgen นั้นจะประกอบไปด้วย procedure ชื่อเดียวกับที่ได้ประกาศไว้ในไฟล์ header นั่นเอง ซึ่งจะเป็นส่วนที่รับการเรียก procedure ปกติจากผู้เรียก แล้วนำ parameter ส่งไปยัง stub ของผู้ให้บริการ โดยใช้ XDR filter ในการแปลงรูปแบบข้อมูล และรอรับ return value จาก ผู้ให้บริการเพื่อทำการส่งกลับไปให้แก่ผู้เรียกอีกทีหนึ่ง โดยที่ผู้เรียกจะรู้สึกเหมือนกับการเรียกใช้ local procedure ธรรมดาเท่านั้น

ส่วน stub ของผู้ให้บริการนั้นจะมีฟังก์ชัน main() ซึ่งจะทำหน้าที่ register โปรแกรมแล้วรอให้มีผู้เรียก procedure ที่มีอยู่ แล้วจึงทำการแปลง parameter ที่ได้รับในรูป XDR กลับให้อยู่ในรูปเดิม ทำการเรียก procedure ที่ต้องการ และ ส่ง return argument กลับไปให้แก่ผู้เรียกในรูป XDR อีกทีหนึ่ง

2.2.2.2) การเขียน application code ส่วนผู้เรียก และ ผู้ให้บริการ

สำหรับ code ที่ผู้ใช้จะต้องเขียนเอง ในส่วนผู้เรียกนั้นคือฟังก์ชัน main() และ local procedure อื่นๆ โดยเมื่อต้องการจะเรียก remote procedure นั้น ก่อนอื่นจะต้องทำการสร้างการติดต่อกับเครื่องที่มีผู้ให้บริการวิ่งอยู่ และ ทำการขอ register เพื่อตรวจสอบดูว่ามี procedure ที่ต้องการหรือไม่ (เปรียบเสมือนการขอเปิดไฟล์ เพื่อทำการ อ่าน/เขียน) โดยการเรียกฟังก์ชัน

clnt_create(3N) ซึ่งถ้าไม่มีอะไรผิดพลาดก็จะ ได้รับ client handler กลับมา(เปรียบเสมือนกับไฟล์ handler) ซึ่งจะต้องใช้ในการเรียก remote procedure ต่อไป

เมื่อได้ client handler มาแล้วก็จะสามารถเรียก remote procedure ได้เหมือนกับการเรียก local procedure ปรกติเพียงแต่ต้องใส่ client handler เป็น parameter เพิ่มเติมจาก parameter ที่กำหนดไว้เดิม โดยที่การส่ง parameter ทุกตัวนั้นจะเป็นการส่ง address ของตัวแปรไป และ return value ที่ได้ก็จะเป็น address ของ return value นั้นๆ

ก่อนที่จะจบโปรแกรมก็จะต้องทำการ เลิกการติดต่อกับผู้ให้บริการ(เปรียบเสมือนกับการปิดไฟล์) โดยการทำลาย client handler ด้วยฟังก์ชัน clnt_destroy(3N)

สำหรับ code ในส่วนของผู้ให้บริการที่จะต้องเขียนเองนั้นก็คือส่วนของ procedure ต่างๆที่ได้กำหนดไว้ในโปรโตคอล แล้วนั่นเอง โดยที่จะใช้ชื่อ ไฟล์ว่า rdb_proc.c หรือ rdb_svc_proc.c

2.2.2.3) การ compile และ ทดสอบโปรแกรม

เมื่อได้ code ครบแล้วก็จะทำการ compile ส่วนของผู้เรียก และ ผู้ให้บริการ แล้ว link กับ stub และ XDR filter เพื่อให้ได้ executable program สอง ไฟล์ คือ *rdb* และ *rdb_svc* ดังตัวอย่างต่อไปนี้

compile client: $ *cc -c -o rdb.o -g rdb.c*

compile client stub: $ *cc -c rdb_clnt.c*

compile XDR filter: $ *cc -c rdb_xdr.c*

link to make client program: $ *cc -o rdb rdb.o rdb_clnt.o rdb_xdr.o*

compile service procedure: $ *cc -c -o rdb_proc.o -g rdb_proc.c*

compile server stub: $ *cc -c rdb_svc.c*

link to make server program: $ *cc -o rdb_svc rdb_proc.o rdb_svc.o rdb_xdr.o*

ทำการทดสอบโดยการ run โปรเซสผู้ให้บริการบนเครื่องใดเครื่องหนึ่งในเครือข่าย

$ *rdb_svc &*

แล้วทำการ run โปรเซสผู้เรียกที่เครื่องอื่นๆ แล้วตรวจดูความถูกต้องของผลที่ได้

```
$ rdb cortex 2  BLOOMER

first     middle last          phone location

JOHN J               BLOOMER    6964   KWC317
```

สำหรับความสามารถอื่นๆ วิธีการเขียนโปรแกรมติดต่อผ่านเครือข่ายขั้นสูง และตัวอย่างการนำ RPC ไปประยุกต์ใช้ในด้านต่างๆนั้น สามารถศึกษาเพิ่มเติมได้จากหนังสืออ้างอิง [8] และ [12] ที่ได้ระบุไว้ท้ายเล่มต่อไป

2.3 ระบบ X Window

2.3.1 X Window คืออะไร

X Window หรือที่เรียกสั้น ว่า X เป็นระบบวินโดว์ที่ไม่ขึ้นกับ Hardware และระบบปฏิบัติการ (Operating System Independent) ที่มีการแสดงผลเป็นแบบบิทแมปกราฟิก (Bitmap Graphic) ไม่ว่าจะเป็นรูปภาพ หรือตัวอักษร (Text) ต่างๆ ระบบ X Window มีลักษณะคล้ายกับระบบวินโดว์อื่นๆ ทั่วไปคือ เใช้ลักษณะการแสดงผลที่มีการแบ่งจอภาพออกเป็นส่วนๆ ที่เรียกว่า วินโดว์ (Window) โดยแต่ละส่วนจะมีส่วนรับข้อมูลและส่วนแสดงผล (Input and Output) เป็นของตนเอง โดยส่วนใหญ่แล้วส่วนรับข้อมูลจะเป็นแป้นพิมพ์ (Keyboard) และตัวชี้ (Pointer) หรือ mouse นั่นเอง

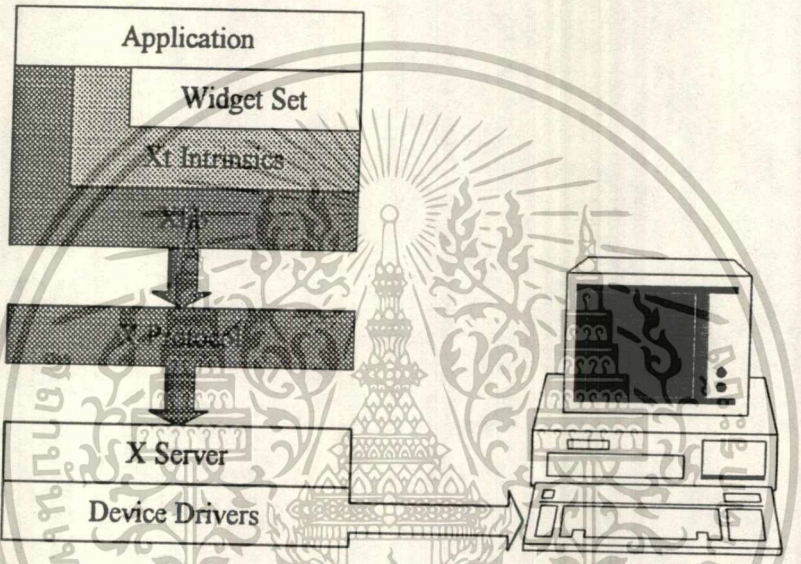
X Window เป็นระบบลูกข่ายกับผู้ให้บริการ (Client - Server) คือการที่สามารถให้โปรแกรมวิ่ง (Run) หรือทำงานอยู่บนเครื่องๆ หนึ่ง และแสดงผลที่หน้าจอ (Terminal) ของเครื่องอีกเครื่องหนึ่ง ซึ่งอาจเป็นหน้าจอของเครื่องที่วิ่งโปรแกรมอยู่ก็ได้ โดยหน้าจอส่วนที่แสดงผลนั้นเรียกว่า Display Server และโปรแกรมที่วิ่งอยู่บนอีกเครื่องเรียกว่า Client หรือ Application ระบบ X Window นี้อนุญาตให้มีการใช้งานเชื่อมต่อกับเครื่องลูกข่ายหลายๆ ตัวได้บนหน้าจอผู้ให้บริการเดียว เช่นอาจมีการเชื่อมต่อกับเครื่องเมนเฟรม (Main Frame) ที่วินโดว์หนึ่ง และมีการต่อกับเครื่อง Workstation ที่อีกวินโดว์หนึ่งก็ได้ ข้อมูล ที่ใช้ส่งไปมาระหว่างลูกข่ายกับผู้ให้บริการนั้นมีชื่อเรียกต่างกัน กล่าวคือ

- ☐ ข้อมูลที่ลูกข่ายส่งไปให้ผู้ให้บริการเรียกว่า Request เช่นมีการเคลื่อนย้ายวินโดว์หรือมีการเปลี่ยนขนาดของวินโดว์ ก็จะส่ง Request ไปบอกให้ตัวผู้ให้บริการวาดหน้าจอใหม่ เป็นต้น หรืออาจจะเป็นการส่งไปเพื่อขอข้อมูลรายละเอียดต่างๆ เกี่ยวกับจอภาพก็ได้
- ☐ ข้อมูลที่ผู้ให้บริการส่งไปให้ลูกข่ายเรียกว่า Event เช่นผู้ใช้มีการกดแป้นพิมพ์ หรือกดปุ่ม หรือลาก mouse ก็จะมีการส่ง Event ไปยังลูกข่ายเพื่อให้ลูกข่ายรับรู้และทำตามคำสั่งที่กำหนดไว้

นอกจากนี้ยังมีการช่วยในการจัดการข้อมูลที่ซับซ้อน เช่นรายละเอียดของวินโดว์ รูปแบบของตัวอักษร (Font) เป็นต้น โดยลูกข่ายจะอ้างถึงเฉพาะหมายเลขประจำตัว (ID) ของข้อมูลชนิดนั้นๆ เท่านั้น ซึ่งจะช่วยลดภาระของระบบเครือข่ายลงได้มาก สำหรับข้อตกลงที่ใช้ในการส่งข้อมูลระหว่างลูกข่ายและผู้ให้บริการนั้น เรียกว่า X Protocol

สำหรับโปรแกรมที่วิ่งบนระบบ X Window นั้นสามารถเขียนได้หลายแบบขึ้นอยู่กับการเลือกชั้นหรือระดับชั้น (Layer) ของ X Library ซึ่งมีดังนี้

- ☐ X Lib
- ☐ Xt Intrinsic
- ☐ Widget Set



รูปที่ 2.3 - 1 สถาปัตยกรรมของโปรแกรมที่เขียนโดยใช้ Xt Intrincsics

2.3.2 Xlib

เป็นชั้นที่อยู่ล่างสุดของทั้ง 3 ชั้น ในการเขียนโปรแกรมเพื่อใช้งานบนระบบ X Window นั้นจะต้องมีการติดต่อ (Interface) กับ X Protocol ซึ่ง X Lib เป็นชั้นที่มีความสามารถและมีฟังก์ชันที่ใช้ติดต่อโดยตรงกับ X Protocol ได้มากที่สุด

ในการติดต่อกับ X Protocol จะต้องมีการส่งแพกเกจของข้อมูล (Packet of Information) แลกเปลี่ยนกันระหว่างลูกข่ายกับผู้ให้บริการ X Lib เป็นชั้นที่มีการสร้าง Protocol Request และส่งไปให้ผู้ให้บริการ โดยใน Protocol Request นั้นจะมีข้อมูลต่างๆ บรรจุอยู่ เช่น สี

ขนาดของเส้น เป็นต้น และ Protocol Reply ซึ่งจะส่งมาจากผู้ให้บริการตอบกลับไปยังลูกข่ายก็สร้างมาจาก X Lib ด้วยเช่นกัน และยังรวมถึง Even กับ Error ด้วย

ตัวจัดการวินโดว์ (Window Manager) ที่เขียนโดยใช้ X Lib มีหน้าที่ในการจัดการทรัพยากร (Resource) ที่ต้องการใช้วินโดว์ทั้งหมด เช่น สี เป็นต้น และเป็นตัวควบคุมการเปลี่ยนแปลงของวินโดว์ เช่นมีการขยายหรือลดขนาดของวินโดว์ การเคลื่อนย้ายวินโดว์ ตัวจัดการวินโดว์นี้จะเป็นตัวทำทั้งหมด

Window

X Server เป็นการแสดงผลกราฟิกแบบบิทแมป (Bitmap) โดยจะมีการแบ่งหน้าจอออกเป็นส่วนๆ เรียกว่า "วินโดว์" โดยวินโดว์แต่ละอันจะใช้ในการรับส่งข้อมูลติดต่อกับเครื่องคอมพิวเตอร์แต่ละเครื่อง (อาจจะติดต่อกับเครื่องเดียวกันหลายๆ ครั้งก็ได้)และใช้ในการแสดงผล วินโดว์แต่ละอันมีขนาดที่แตกต่างกันได้และสามารถย่อหรือขยายขนาด (Resize) ได้ สามารถทำให้เล็กที่สุดที่เรียกว่า "ไอคอน" (Icon) ได้ และสามารถเคลื่อนย้าย (Move) ได้ วินโดว์มีสถานภาพ (Configuration) ต่างๆ ดังนี้

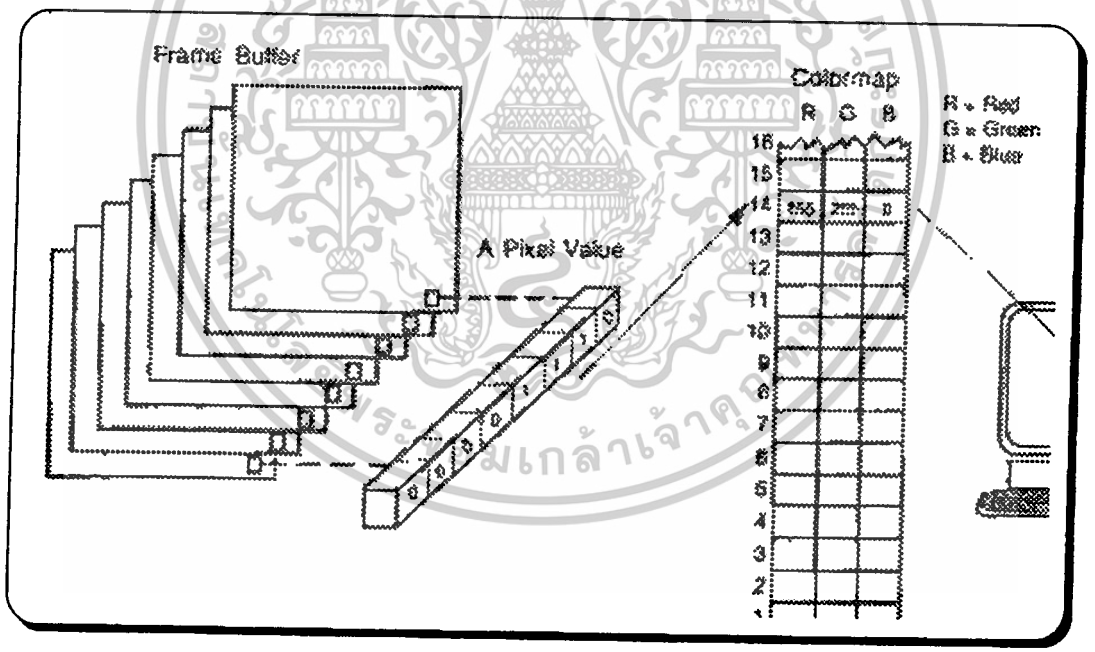
- ☐ ความกว้าง (Width) และความสูง (Height) ของวินโดว์ มีหน่วยเป็นจุด (Pixel) ซึ่งความกว้างและความสูงนี้วัดโดยไม่รวมกรอบ
- ☐ กรอบ (Border) ของวินโดว์ ซึ่งมีขนาดเท่าไรก็ได้ ถ้ามีขนาดเป็นศูนย์ วินโดว์นั้นก็จะไม่มีกรอบ
- ☐ ตำแหน่ง (Position) ของวินโดว์ ใช้สัญลักษณ์ X, Y โดยระบุเป็นจุด ซึ่งจะสัมพันธ์ (Relative) กับวินโดว์ตัวแม่ (Parent Window) โดยตำแหน่ง X, Y นั้นเป็นตำแหน่งซ้ายบนของวินโดว์
- ☐ ลำดับขั้น (Stack Order) ของวินโดว์ ใช้ระบุขั้นของวินโดว์ต่างๆ ภายใต้วินโดว์แม่เดียวกัน ถ้าวินโดว์ไหนอยู่ที่ตำแหน่งบนสุดของลำดับขั้นก็จะปรากฏเป็นวินโดว์ที่อยู่บนสุดในจอภาพ

X Graphics

ระบบกราฟิกของ X นั้นประกอบด้วยส่วนสำคัญต่างๆ ดังนี้

จุดและสี

ระบบ X นั้นถูกออกแบบมาให้ควบคุมระบบแสดงภาพแบบบิทแมป ซึ่งถ้าเป็นจอแสดงผลแบบสีเดียว หรือจอขาวดำนั้น การแสดงจุดจะใช้จำนวนบิท 1 บิทต่อหนึ่งจุด (1 บิทแทนค่าได้ 2 ค่าคือ 0 กับ 1 หรือแทนขาวกับดำนั่นเอง) แต่ถ้าเป็นจอแสดงผลที่สามารถแสดงระดับสีเทา (Gray Scale) หรือจอแสดงผลแบบสีจะใช้บิทหลายบิทต่อจุด 1 จุด แต่การแสดงระดับสีเทาหรือการแสดงสีนั้นจะไม่ใช้การแทนสีด้วยค่าโดยตรง แต่จะใช้วิธีระบุเป็นดรรชนี (Index) โดยดรรชนีนี้จะชี้ไปยังแผนผังสี (Colormap) ตามรูปที่ 2.3 - 2 การแสดงผลในจอสีนั้นใช้วิธีการแยกสีออกเป็น 3 สีคือ แดง เขียว และน้ำเงิน สีที่ปรากฏบนจอนั้นเกิดจากลำอิเลคตรอนของทั้ง 3 สีที่ยิงออกมาบนจอที่จุดเดียวกัน แผนผังสีก็คือตารางที่เก็บค่าของสีแดง เขียว และน้ำเงินนั่นเอง เช่นถ้าระบุค่าของจุดเป็น 29 สีที่แสดงนั้นจะเป็นสีที่เกิดจากการนำสีแดง เขียว น้ำเงินที่เก็บอยู่ในตำแหน่งที่ 29 ในแผนผังสีเป็นต้น จอแสดงผลโดยทั่วไปจะมีแผนผังสี 1 ชุด (อาจมี 2 ชุดก็ได้ขึ้นอยู่กับชนิดของจอ) ดังนั้นส่วนใหญ่จะมีการทำแผนผังสีเสมือน (Virtual Colormap) เพื่อให้ใช้แผนผังสีได้หลายแบบ



รูปที่ 2.3-2 การใช้แผนผังสี

จุดและระนาบ

จำนวนของบิทต่อจุด นั้นก็คือจำนวนระนาบ (Plane) นั่นเอง สำหรับจอสีเดียวนั้นจะมีจำนวนระนาบ เท่ากับ 1 แต่สำหรับจอที่สามารถแสดงระดับของสีเทาและจอสีนั้นจะมีจำนวนระนาบได้ตั้งแต่ 4 ถึง 28 ระนาบ และ X นั้นได้ออกแบบให้รองรับได้ถึง 32 ระนาบเลยทีเดียว สำหรับจอสีที่มี 4 ระนาบนั้นจะสามารถแสดงสีได้ถึง 16 สี ถ้ามี 8 ระนาบจะแสดงสีได้ 256 สี จะเห็นว่าจำนวนสีนั้นคำนวณได้จาก 2^n เมื่อ n คือจำนวนระนาบ สำหรับจำนวนระนาบนี้เรียกอีกอย่างได้ว่า "ความลึก" (Depth)

ในการคำนวณหาตำแหน่งของจุดสีและการระบายจุดสีนั้น ต้องคำนวณโดยใช้องค์ประกอบหลายอย่าง เช่นสีเดิม, ตำแหน่งเดิม, การระบุระนาบ (Plane Mask) การระบุขอบ (Clip Mask) และฟังก์ชันทางตรรกต่างๆ (Logical Function) ซึ่งข้อมูลพวกนี้นั้น X ได้มีการกำหนดไว้ในโครงสร้างข้อมูล (Data Structure) ที่เรียกว่า "สภาวะของกราฟิก" (Graphic Context, GC)

Pixmap และ Drawable

ในการวาดภาพนั้นไม่จำเป็นจะต้องวาดลงบนวินโดว์เท่านั้น อาจจะวาดลงบนพิกแมป (Pixmap) ก็ได้ ซึ่งพิกแมปกับวินโดว์นี้เราเรียกว่า Drawable

พิกแมปเป็นอาร์เรย์ของค่าของจุด (Pixel Value) ที่มีความลึกเหมือนกับวินโดว์ แต่ว่าจะไม่มีตำแหน่งที่สัมพันธ์กับวินโดว์หรือพิกแมปอื่นๆ และไม่มีคุณลักษณะ (Attribute) ของวินโดว์เช่นแผนผังสี เป็นต้น พิกแมปจะเป็นตัวที่เก็บพวกรูปกราฟิกทั้งหลายไว้ในหน่วยความจำ แต่จะไม่แสดงออกมาบนจอภาพ จนกว่าจะมีการคัดลอก (Copy) จากพิกแมปไปยังวินโดว์ที่มองเห็นได้ (Visible) และสำหรับพิกแมปที่มีความลึก 1 นั้นจะเรียกว่า " บิทแมป "

Event

Event คือเหตุการณ์ที่ปรากฏขึ้นเมื่อมีการกระทำ (Action) ต่างๆ เช่นถ้าเราเลื่อน Mouse หรือกดแป้นพิมพ์ ก็จะมี Input Event เกิดขึ้น สำหรับ Event ตัวอย่าง 2 อันนี้เป็น Event แบบง่ายๆ ยังมี Event ที่ซับซ้อนกว่านี้อีก สำหรับโครงสร้างข้อมูลของ Event นี้จะเป็นแพกเกตของข้อมูลที่สร้างที่ผู้ให้บริการ เมื่อมีการกระทำต่างๆ เกิดขึ้น และจะเก็บไว้ในคิว (Queue) เพื่อรอที่จะส่งไปให้ลูกข่าย

Event Handler

ในการควบคุม Event ของโปรแกรมมีขั้นต้อน 3 ขั้นตอนดังนี้

- ขั้นแรกโปรแกรมจะต้องเลือก Event ที่ต้องการสำหรับแต่ละวินโดว์
- ขั้นที่สองจัดการ Map วินโดว์นั้น
- ขั้นที่สามปล่อยให้เป็นหน้าที่ของ Event Loop ที่จะอ่าน Event จากคิวเวลาที่มี Event เกิดขึ้น

โครงสร้างของ Event นั้นประกอบไปด้วยแพกเกตของข้อมูลชนิดต่างๆ สำหรับโครงสร้างของ Event แบบง่ายๆ นั้นมีดังนี้

```
typedef struct {
        int       type;          /* the type of event */
        unsigned long serial;    /* # of last request processed by server */
        Bool send_event;         /* true if send from a SendEvent request */
        Display *display;        /* display the event was read from */
        Window window;           /* window that receives event */
} XAnyEvent;
```

และสำหรับ Event แบบอื่นๆ นั้นก็จะคลายๆ กันเนื่องจากโครงสร้างของตัว XEvent นั้น เป็นแบบ Union ดังนี้

```
typedef union _XEvent {
        int             type;
        XAnyEvent       xany;
        XKeyEvent       xkey;
        XButtonEvent    xbutton;
        XMotionEvent    xmotion;
        XCrossingEvent  xcrossing;
```

```
    XFocusChangeEvent     xfocus;
    XKeymap               xkeymap;
    XExposeEvent          xexpose;
    XNoExposeEvent        xnoexpose;
    .
    .
    .
    XColormapEvent        xcolormap;
    XClientMessageEvent   xclient;
} XEvent;
```

2.3.3 Xt Intrinsic และ Widget Set

สำหรับโปรแกรมเมอร์มือใหม่หรือผู้ที่ไม่มีความชำนาญนั้น อาจจะเป็นเรื่องยากที่จะต้องสร้างโปรแกรมที่มีการติดต่อกับผู้ใช้แบบกราฟิก (Graphic User Interface, GUI) โดยใช้ภาษาระดับล่าง (Low Level) อย่าง X Lib เป็นต้น ดังนั้นจึงได้มีการออกแบบและสร้างฟังก์ชันต่างๆ ที่จำเป็นในการสร้าง GUI พวก Scroll Bar, ปุ่ม (Button) ต่างๆ, Dialog Box, Popup หรือ เมนูแบบพูลดาวน์ (Pulldown Menu) ฯลฯ ขึ้นมา

จุดประสงค์ของ X Toolkit นั้นก็เพื่อให้การเขียนโปรแกรมจำพวก GUI ให้ง่ายขึ้น โดยลักษณะของโปรแกรมนั้นจะมีลักษณะที่ง่ายต่อการสร้างขึ้นมา แต่จะไม่จำกัดแบบและรูปร่าง โดยโปรแกรมเมอร์สามารถแก้ไขเพิ่มเติมได้ หรือที่เรียกว่าการนำกลับมาใช้ใหม่ (Reuseable) สำหรับ Xt นี้จะมีลักษณะการโปรแกรมเป็นแบบวัตถุ (Object - Oriented style) โดยมีพื้นฐานเป็นภาษาซีแต่เรียกใช้ Library ที่เรียกว่า Xt หรือ X Toolkit Intrinsic นี้เอง Xt มีฟังก์ชันและ Routine ต่างๆ ที่ใช้สำหรับสร้างส่วนที่จะต้องติดต่อกับผู้ใช้ที่เรียกว่า "วิดเจต" (Widget) ซึ่งวิดเจตนี้มีอยู่ 2 แบบใหญ่ๆ คือ Motif กับ Open Look แต่สำหรับโปรแกรมนี้เราจะกล่าวถึงและใช้เฉพาะวิดเจต ของ Motif เท่านั้น สำหรับวิดเจตที่ใช้โดยทั่วไปก็ได้แก่

- ☐ Push Button หรือ Command Button มีหลักการดังนี้คือ จะทำงานเมื่อถูกกด
- ☐ Scroll Bar สำหรับเลื่อนเพื่อดูรูปหรือตัวอักษรที่อยู่ถัดไปที่มองไม่เห็นทั้งหมด
- ☐ Data Area สำหรับรับข้อมูลที่พิมพ์เข้ามา เป็นต้น

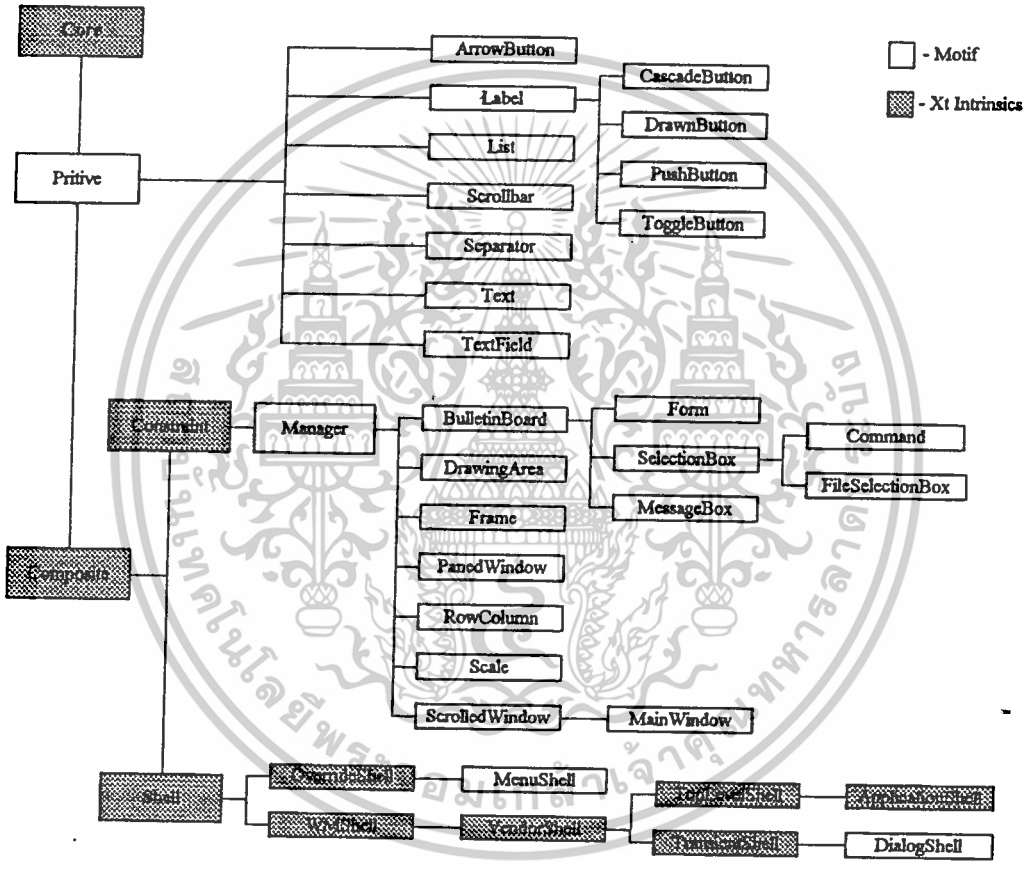
และนอกจากนี้ยังมีวิดเจตแบบอื่นๆ และแบบที่ไม่สามารถมองเห็นได้ เช่น

- ☐ Composite Widget
- ☐ Shell Widget
- ☐ Popup Dialog Box
- ☐ Special Purpose Application Window

Widget Class

หลักการของวิดเจตแต่ละชนิดก็คือ จะมีลักษณะต่างๆ ที่สืบทอด (Inherit) มาจากวิดเจตที่เป็น Super Class ของมัน เช่น Arrow Button นั้นมีการสืบทอดมาจาก Primitive Widget เป็นต้น ในการสร้างวิดเจตแต่ละครั้งนั้น เราจะสร้างเฉพาะ Instance ของมันเท่านั้น

โดย วิดเจตแต่ละชนิดนั้นจะถูกออกแบบไว้ก่อนแล้วและเรียกรวมๆ กันว่า "วิดเจตคลาส" (Widget Class) เมื่อเราสร้างวิดเจตแต่ละอันจะหมายถึงการกำหนดรายละเอียดต่างๆ ของวิดเจตชนิดนั้นๆ (วิดเจตชนิดต่างๆ จะมีรายละเอียดที่ถูกกำหนดไว้แล้วแต่ยังไม่ได้กำหนดค่าไว้) ดังนั้นวิดเจต ชนิดเดียวกันก็อาจจะมีรูปร่างที่ไม่เหมือนกันก็ได้ขึ้นอยู่กับค่ารายละเอียดต่างๆ ที่เราตั้งไว้ รายละเอียดต่างๆ ได้แก่ ตำแหน่ง, ความกว้าง, ความสูง เป็นต้น สำหรับคลาส (Class) ต่างๆ ที่กำหนดไว้มีดังรูปที่ 2.3 - 3



รูปที่ 2.3-3 แสดงลำดับคลาสของ Motif Widget Set

Xt และการโปรแกรมเชิงวัตถุ (OOP)

วัตถุ (The Object)

ใน OOP นั้นวัตถุจะประกอบไปด้วย 2 ส่วนคือ ส่วนข้อมูล (Data) และวิธีการ (Method) สำหรับอ่านและเขียนข้อมูลนั้นๆ ซึ่งข้อมูลนี้จะมีข้อมูลแบบส่วนตัว (Private) คือวัตถุอื่นจะไม่สามารถมองเห็นข้อมูลนี้ได้ กับข้อมูลแบบสาธารณะ (Public) คือวัตถุอื่นสามารถมองเห็นได้

วิธีการ (Method)

วิธีการก็คือกลุ่มของฟังก์ชันที่กำหนดไว้สำหรับคลาสแต่ละคลาสโดยเฉพาะสำหรับ วิดเจตนั้นๆ ตัวอย่างวิธีการอย่างง่ายๆ ก็เช่น ฟังก์ชันการสร้างและทำลายวินโดว์ เป็นต้น แต่วิธีการเหล่านี้โปรแกรมเมอร์สามารถกำหนดหรือเขียนขึ้นมาใหม่ตามที่ต้องการได้ ซึ่งในวิธีการเหล่านี้ของแต่ละวินโดว์นั้น จะมีชุดของ Argument ที่เหมือนกันทำให้เขียนโปรแกรมง่ายขึ้น

ข่าวสาร (Message)

สำหรับภาษาที่เป็น OOP อย่างแท้จริง การที่จะติดต่อสื่อสารกันระหว่างแต่ละวัตถุนั้น จะใช้วิธีการส่งข่าวสารไปมาระหว่างแต่ละวัตถุ ซึ่งในระบบ X Window นั้นข่าวสารที่ส่งจะอยู่ในรูปของ Event, Request นั่นเอง

การเขียนโปรแกรมโดยใช้ Xt Intrinsic

ในการเขียนโปรแกรมโดยใช้ Xt นั้นที่จริงแล้วก็คือการเขียนโปรแกรมด้วยภาษาซีธรรมดาโดยเรียกใช้ฟังก์ชันไลบรารีของ Xt นั่นเอง เช่นการสร้างวิดเจตใหม่โดยอาศัยวิดเจตที่มีอยู่แล้วเป็นฐาน สำหรับวิดเจตที่เราจะสร้างและนำมาประกอบกันเป็นวิดเจตใหม่นั้นอย่างน้อยที่สุดจะต้องเป็น Composite Widget คือวิดเจตที่สามารถประกอบกันเป็นวิดเจตอื่นได้ การสร้างวิดเจตจะต้องสร้างโดยเรียงลำดับให้ถูกต้อง โดยต้องสร้างวิดเจตที่เป็นซุปเปอร์คลาส (Super Class) หรือวิดเจตแม่ (Parent Widget) ก่อนแล้วจึงสร้างวิดเจตที่อยู่ในคลาสถัดๆ มา ตัวอย่างเช่นถ้าเราจะสร้างวินโดว์ที่มีเมนูแบบพูลดาวน์ ก็จะมีขั้นตอนในการทำดังนี้

- ☐ ขั้นแรกก็จะต้องสร้างวินโดว์อย่างง่ายๆ (Simple Window) ขึ้นมาก่อน
- ☐ สร้าง Bulletin Board Widget เป็นลูกของวินโดว์
- ☐ สร้าง Menu Bar Widget มาเป็นลูกของ Bulletin Board Widget
- ☐ สร้าง Pulldown Menu Widget มาเป็นลูกของ Menu Bar Widget อีกที
- ☐ สุดท้ายสร้าง Push Button Widget มาเป็นลูกของ Pulldown Menu Widget

ก็จะได้วินโดว์อย่างเรียบๆ ที่มีพูลดาวน์เมนูหนึ่งอัน สำหรับการสร้างโปรแกรมอย่างง่ายๆ ขึ้นมาสักชิ้นหนึ่งนั้นมีหลักการดังนี้

- ☐ กำหนดค่าตัวแปรต่างๆ ให้ครบ เช่นชื่อวิดเจตชนิดต่างๆ

- ☐ กำหนดค่าเริ่มต้น (Initialize) ให้กับตัวแปรและสภาวะของโปรแกรม (Application Context) ให้ครบ
- ☐ วางโครงสร้างของวินโดว์หรือวิดเจตที่ต้องการ โดยกำหนดว่าต้องการวิดเจตอะไรบ้าง และวิดเจตแต่ละอันนั้นต้องมีวิดเจตแม่อะไรบ้าง, ต้องมีสภาวะของวิดเจตอะไรบ้าง
- ☐ สร้างวิดเจตที่ต้องการตามลำดับขั้น โดยสร้างวิดเจตที่อยู่ในคลาสที่ใหญ่ที่สุดก่อน เช่น Shell Widget Class เป็นต้น (แต่โดยปกติถ้าเราไม่กำหนดค่าเริ่มต้น หรือเปลี่ยนแปลงค่าเริ่มต้นของคลาสเหล่านี้ ตัว Xt จะใช้ค่ามาตรฐานที่กำหนดไว้เป็นค่าเริ่มต้นเอง ซึ่งโปรแกรมที่ไม่ซับซ้อนมากนัก ก็ไม่ควรที่จะไปกำหนดค่าต่างๆ ของวิดเจตที่อยู่ในคลาสใหญ่ๆ เหล่านี้) แล้วค่อยสร้างวิดเจตที่เป็น Sub Class ลงมาเรื่อยๆ
- ☐ สำหรับวิดเจตที่สร้างแต่ละตัวจะต้องมีการกำหนดให้อยู่ในความควบคุมของวิดเจตแม่โดยใช้คำสั่ง XtManageChild
- ☐ ทำการ Realize Widget ที่อยู่ในคลาสใหญ่ที่สุดที่สร้าง
- ☐ เรียกฟังก์ชัน MainLoop เพื่อให้โปรแกรมทำการรอ Event ต่างๆ แล้วจัดการทำงานตาม Event ที่เกิดขึ้นนั้นๆ

และสำหรับวิดเจตที่มีฟังก์ชันที่ต้องทำเวลาที่มี Event เกิดขึ้น เราเรียกฟังก์ชันเหล่านี้ว่า Callback โดย Callback ของแต่ละวิดเจตเป็นฟังก์ชันที่จะทำเวลาวิดเจตนั้นได้รับ Event ที่มันกำหนดไว้ว่าจะรับ และเมื่อทำ Callback เรียบร้อยแล้วการควบคุมก็จะกลับไปอยู่ใน MainLoop เหมือนเดิมเพื่อรอ Event ใหม่

ในการเขียนโปรแกรมโดยใช้วิดเจตนี้ เราสามารถกำหนดค่าของทรัพยากรต่างๆ ของแต่ละวิดเจตไว้ภายนอกโปรแกรมได้ โดยเขียนใส่ไว้ในไฟล์ตัวอักษร (Text File) ที่เรียกว่าไฟล์โปรแกรมมาตรฐาน (Application Default File) ซึ่งในไฟล์นี้เราสามารถกำหนดค่าทรัพยากรต่างๆ เช่น ตำแหน่งของวินโดว์, สี, ขนาดความกว้างความสูง และค่าอื่นๆ ได้อย่างอิสระ ซึ่งถ้าเราเปลี่ยนแปลงค่าเหล่านี้ก็จะทำให้รูปร่างหน้าตาของโปรแกรมเปลี่ยนไปด้วย (แต่ในโครงงานนี้จะไม่ได้ใช้ไฟล์มาตรฐานนี้ในการกำหนดค่าต่างๆ แต่จะกำหนดไว้ในโปรแกรมเลย เพื่อไม่ให้มีการเปลี่ยนแปลงรูปร่างและสภาวะบางอย่างของโปรแกรม)

การควบคุม Event (Event Handling)

สำหรับการควบคุม Event ต่างๆ นั้นจะคล้ายๆ กับใน X Lib แต่จะเป็นชื่อฟังก์ชันที่มีอยู่ใน Xt แทน ซึ่งจริงๆ แล้วบางฟังก์ชันก็เป็นเพียงมาโครเพื่อไปเรียกฟังก์ชันของ X Lib ที่มีอยู่แล้วนั่นเอง

การควบคุมแป้นพิมพ์และ Mouse

การควบคุมแป้นพิมพ์และ Mouse นั้นก็เป็นฟังก์ชันที่ทำงานเมื่อมี Event เกิดขึ้นนั่นเอง โดยใช้การ Add Callback ฟังก์ชันลงไปในวิดเจตที่ต้องการรับข้อมูลทางแป้นพิมพ์ กับ Mouse นั้นๆ เช่น Push Button เมื่อเรากดแป้นพิมพ์ หรือใช้ Mouse คลิกไปตรง Push Button ก็จะได้รับ Event (KeyPress ถ้ากดแป้นพิมพ์หรือ ButtonPress ถ้าเป็นการกดปุ่ม) ฟังก์ชัน Callback ก็จะทำงาน

สำหรับรายละเอียดวิธีการพัฒนาโปรแกรมบน X Windows ด้วยวิธีการต่างๆนั้น สามารถศึกษาเพิ่มเติมได้จากหนังสืออ้างอิง [2], [3], [4], [5], [6], [7], [9], [10] และ [13] ที่ได้ระบุไว้ท้ายเล่มต่อไป

2.4 การติดต่อระหว่างโปรเซส (Interprocess Communication)

ในการควบคุมการเข้าจังหวะระหว่างแต่ละโปรเซสที่มีการทำงานขนานกัน และต้องการที่จะติดต่อกัน หรือใช้ข้อมูลร่วมกันนั้นจะต้องมีการควบคุมการเรับ-ส่งและการข้าถึงข้อมูลของแต่ละโปรเซสให้ถูกต้อง สำหรับโครงงานนี้วิธีการควบคุมที่ใช้คือ "เซมาฟอร์" (Semaphore)

2.4.1 Share Memory

ในระบบลูกข่ายกับผู้ให้บริการนั้น วิธีการส่งข้อมูลให้กันและกันนั้นมีได้หลายวิธี เช่นการส่งโดยใช้วิธีผ่านข้อความ (Message Passing), การใช้ FIFO pipeและการใช้หน่วยความจำร่วมกัน (Share Memory) ซึ่งวิธีการใช้หน่วยความจำร่วมกันนี้คือการที่โปรเซสตั้งแต่ 2 โปรเซสขึ้นไปมีการใช้ตัวแปรหรือข้อมูลชุดเดียวกัน เนื่องจากข้อมูลที่แต่ละโปรเซสใช้นั้นเป็นข้อมูลชุดเดียวกัน ดังนั้นการที่โปรเซสใดเขียนข้อมูลทับลงไป และอีกโปรเซสมีการอ่านข้อมูลนั้นขึ้นมา ก็คือการส่งข้อมูลผ่านไปมาระหว่างโปรเซสนั่นเอง แต่เนื่องจากว่ามีการเข้าถึงหน่วยความจำส่วนนี้มากกว่าหนึ่งโปรเซส ดังนั้นจะต้องมีการจัดการการป้องกันความเสียหายที่จะเกิดขึ้นจากการที่โปรเซสเหล่านี้ใช้หน่วยความจำพร้อมๆ กัน ซึ่งวิธีการที่ว่านี้ก็คือการใช้เซมาฟอร์นั่นเอง

ในการทำการใช้หน่วยความจำร่วมกันนั้น เราจะต้องมีตัวเลขชุดหนึ่งสำหรับเป็นตัวบอกให้แต่ละโปรเซสรู้ว่าเป็นหน่วยความจำที่ต้องการจะใช้ร่วมกัน เราเรียกตัวเลขนี้ว่า "หมายเลขประจำตัวของหน่วยความจำร่วมกัน (shmid) คำสั่งที่ใช้กับหน่วยความจำร่วมกันนี้มีดังนี้

- ☐ shmget เป็นการขอจองที่ไว้สำหรับทำเป็นหน่วยความจำร่วมกัน
- ☐ shmat เป็นการที่โปรเซสที่ต้องการใช้หน่วยความจำร่วมกันนี้ขออนุญาตใช้
- ☐ shmdt เป็นการยกเลิกการขอใช้หน่วยความจำร่วมกัน

2.4.2 เซมาฟอร์ (Semaphore)

เป็นหลักเบื้องต้นของการเข้าจังหวะ โดยอาศัยหลักการอนุญาตให้โปรเซสมีการเข้าถึงทรัพยากรได้หรือไม่ โดยถ้ามีโปรเซสไหนกำลังใช้ทรัพยากรนั้นอยู่ หรือที่เรียกว่าอยู่ในช่วงวิกฤต (Critical Section) ก็จะต้องป้องกันไม่ให้โปรเซสอื่นเข้ามาใช้ทรัพยากรในช่วงวิกฤตนี้ นอกจากนี้ยังใช้สำหรับบังคับให้โปรเซสแต่ละโปรเซสต้องมีการเข้าถึงทรัพยากรตามลำดับ เช่น กำหนดให้โปรเซสแรกต้องเข้ามาจัดการทรัพยากรนั้นก่อน แล้วจึงอนุญาตให้อีกโปรเซสเข้ามาใช้ได้ เป็นต้น

ในการใช้งานนั้นได้ออกแบบเซมาฟอร์โดยใช้เลขจำนวนเต็ม S ค่าหนึ่งเป็นเซมาฟอร์ และมีคำสั่งที่สามารถใช้ได้ 3 อย่างเท่านั้นคือ

- Initial เป็นการเซตค่าเริ่มต้นของเซมาฟอร์นั้นว่ามีทรัพยากรที่อนุญาตให้ใช้เท่าใด
- Wait(S) จะทำการลดค่าของ S ลงหนึ่ง แล้วตรวจดูว่าค่าของ S น้อยกว่า 0 หรือไม่ ถ้าน้อยกว่า โปรเซสที่เรียก Wait ก็จะต้องรอโดยเข้าไปรออยู่ในคิว และถ้ามากกว่าหรือเท่ากับ 0 โปรเซสนั้นก็สามารถใช้งานทรัพยากรนั้นได้
- Signal(S) ทำการเพิ่มค่าของ S ขึ้นหนึ่ง แล้วตรวจดูว่าค่าของ S มากกว่าหรือเท่ากับ 0 หรือไม่ ถ้าใช่ก็จะไปปลุก (Wake Up) โปรเซสแรกที่รออยู่ในคิวออกมาทำงาน

สำหรับโครงงานนี้ได้ใช้วิธีการสร้างเซตของเซมาฟอร์เพื่อควบคุมการเข้าถึงตัวแปรร่วม (shared-variable) โดยในเซตนี้จะประกอบไปด้วยสมาชิก 3 ตัวคือ

- สมาชิกตัวแรก เป็นค่าของเซมาฟอร์ที่แท้จริง ซึ่งผู้ใช้สามารถทำการกำหนดค่าเริ่มต้น, เพิ่มค่า และลดค่าได้
- สมาชิกตัวที่ 2 เป็นจำนวนโปรเซสที่กำลังใช้เซมาฟอร์นี้อยู่ ใช้เพื่อตรวจดูว่ามีโปรเซสไหนใช้เซมาฟอร์ตัวนี้อยู่หรือไม่ และถ้าไม่มีใครใช้อยู่ก็สามารถลบเซตของเซมาฟอร์นี้ทิ้งได้
- สมาชิกตัวที่ 3 เป็นตัวแปรที่ใช้สำหรับล็อก (Lock) ตัวแปร S เพื่อป้องกันเหตุการณ์ที่เรียกว่า Race Condition คือการที่โปรเซสแต่ละโปรเซสพยายามแย่งกันเพื่อที่จะเข้าถึงเซมาฟอร์ เช่นมีโปรเซสหนึ่งกำลังจะตั้งค่าเริ่มต้น และก็มีอีกโปรเซสหนึ่งพยายามที่จะลดค่าเซมาฟอร์นั้น

สำหรับคำสั่งที่ใช้กับเซตของเซมาฟอร์นี้มีดังนี้

- sem_create สำหรับสร้างตัวแปรเซมาฟอร์ขึ้นมาใหม่
- sem_open สำหรับขอใช้ตัวแปรเซมาฟอร์ที่มีการสร้างขึ้นมาก่อนหน้าแล้ว
- sem_rm สำหรับลบตัวแปรเซมาฟอร์ทิ้งไป
- sem_close สำหรับยกเลิกการใช้เซมาฟอร์นั้น และจะมีการตรวจดูว่ามีโปรเซสอื่นยังใช้อยู่อีกหรือไม่ ถ้าไม่มีก็จะเรียก sem_rm เพื่อลบเซมาฟอร์นั้นทิ้งไป

- ☐ sem_wait ลดค่าของเซมาฟอร์ลงหนึ่ง และตรวจดูว่าค่าของเซมาฟอร์นั้นน้อยกว่า 0 หรือไม่ ถ้าน้อยกว่าโปรเซสนั้นก็จะต้องไปรอในคิว
- ☐ sem_signal เพิ่มค่าของเซมาฟอร์ขึ้นหนึ่ง และตรวจดูว่าค่าของเซมาฟอร์นั้นมากกว่าหรือเท่ากับ 0 หรือไม่ ถ้าใช่ก็จะไปปลุกโปรเซสแรกที่รออยู่ในคิวออกมาทำงาน
- ☐ sem_op เป็นการเซตค่าต่างๆ ในเซตของเซมาฟอร์ เพื่อให้เป็นคำสั่งต่างๆ

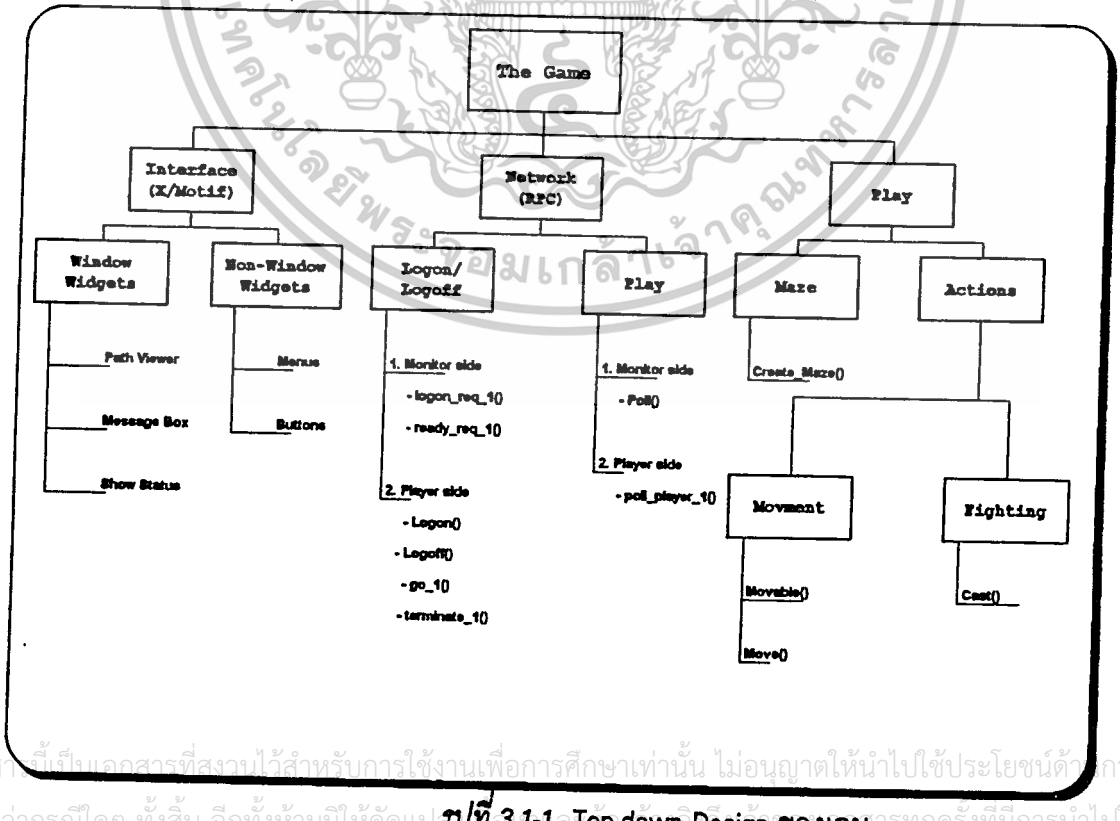
สำหรับตัวอย่างวิธีการใช้ shared-memory และ semaphores และการนำไปประยุกต์ใช้ในด้านต่างๆนั้น สามารถศึกษาเพิ่มเติมได้จากหนังสืออ้างอิง [12] ที่ได้ระบุไว้ท้ายเล่มต่อไป

บทที่ 3
การออกแบบและโครงสร้างของเกม

3.1 โครงสร้างเกม

โครงสร้างทั้งหมดของเกมสามารถแสดงอยู่ในรูปของการออกแบบแบบ top-down design ได้ดังรูป 3-1 โดยจะเห็นได้ว่าเกมถูกแบ่งออกเป็นสามส่วนสำคัญๆดังนี้คือ

1. ส่วนที่ทำหน้าที่เกี่ยวกับการติดต่อกับผู้เล่น (Interfacing) คือส่วนที่ทำหน้าที่รับอินพุท (input) จากผู้เล่นเพื่อนำไปควบคุมการเล่น และ แสดงเอาท์พุท (output) ออกทางจอภาพตามสถานะที่เป็นอยู่ ซึ่งใช้ระบบการติดต่อกับผู้ใช้แบบ X Windows
2. ส่วนที่ทำหน้าที่เกี่ยวกับการติดต่อระหว่างผู้เล่น กับ มอนิเตอร์ (Networking) ซึ่งผู้เล่นแต่ละคนจะมีการสร้างการติดต่อกับมอนิเตอร์อยู่สองส่วนด้วยกัน ได้แก่ ส่วนที่ผู้เล่นใช้ในการขอ Logon/Logoff เพื่อการเริ่ม และ เลิกเล่นเกม กับ ส่วนที่มอนิเตอร์ใช้ในการ poll ผู้เล่นเพื่อส่งสถานะของผู้เล่นอื่นมาให้ พร้อมทั้งรับสถานะปัจจุบันของผู้เล่นกลับไป ซึ่งใช้ RPC ในการติดต่อระหว่างเครื่อง และ ใช้ Shared-memory ร่วมกับ Semaphore ในการติดต่อระหว่างโปรเซสบนเครื่องเดียวกัน

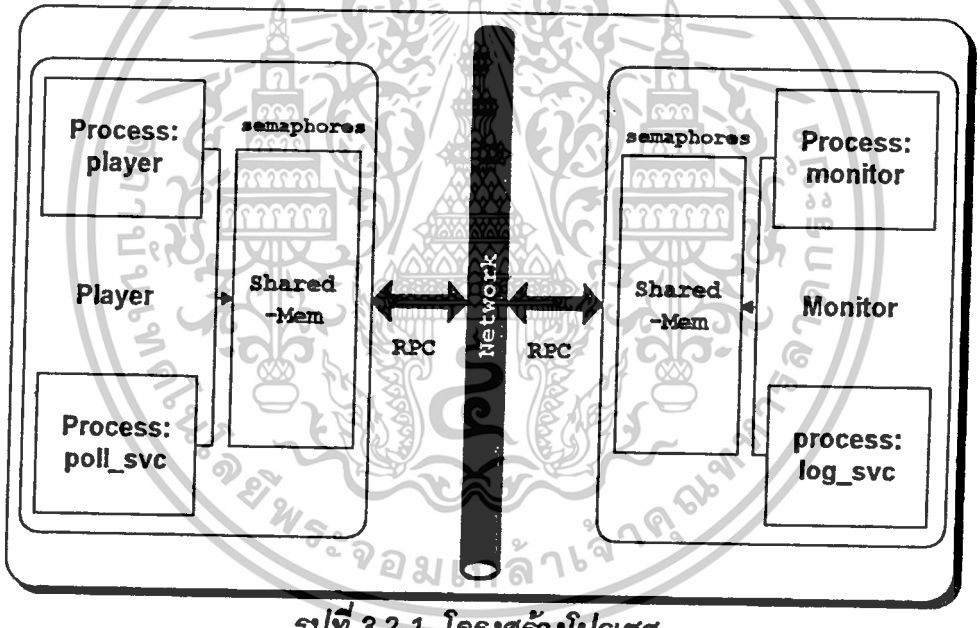


รูปที่ 3.1-1 Top-down Design ของเกม

3. ส่วนที่ทำหน้าที่เกี่ยวกับการเล่น (Playing) คือส่วนที่ทำหน้าที่ในการควบคุมการเปลี่ยนสถานะของผู้เล่นตามที่ได้รับจากผู้เล่นในส่วนที่ (1) และ จากมอนิเตอร์ในส่วนที่ (2) เช่นการ เคลื่อนที่ หรือ เปลี่ยนทิศทาง การปล่อยอาวุธ/เวทมนต์ หรือ การถูกอาวุธ/เวทมนต์จากผู้เล่นอื่นๆในการต่อสู้ เป็นต้น

3.2 โครงสร้างโปรเซส

สำหรับโครงสร้างของโปรเซสในฝั่งมอนิเตอร์ และ ฝั่งผู้เล่นนั้น สามารถแสดงได้ดังรูปที่ 3-2 ซึ่งฝั่งมอนิเตอร์นั้นจะมีสองโปรเซสคือ 'monitor' และ 'logon_svc' ส่วนฝั่งผู้เล่นนั้นจะมีสองโปรเซสเช่นกันคือ 'player' และ 'poll_svc'



รูปที่ 3.2-1 โครงสร้างโปรเซส

3.3 โครงสร้างข้อมูล (Data Structure)

3.3.1 ข้อมูลสถานะผู้เล่น

ผู้เล่นแต่ละคนจะมีข้อมูลต่างๆที่จะต้องใช้ในการเล่น และ การติดต่อกับมอนิเตอร์โดยจะเก็บไว้ในโครงสร้างข้อมูลแบบ struct ซึ่งถูกชี้โดยอาเรย์ (array) ที่ชื่อว่า Players โดยที่ขนาดของอาเรย์จะเป็นตัวกำหนดจำนวนผู้เล่นสูงสุดที่มีได้ในขณะใดขณะหนึ่งซึ่งกำหนดโดยค่าคงที่ MAX_PLAYERS โดยที่ในโปรแกรมตัวอย่างนี้ได้กำหนดไว้ที่ 4 ผู้เล่น

โครงสร้างข้อมูลที่ใช้เก็บสถานะของผู้เล่นจะประกอบไปด้วยฟิลด์ต่างๆดังนี้

- ☐ phost; ใช้เก็บชื่อ (hostname) ของเครื่องที่ผู้เล่นใช้
- ☐ changed; มีค่าเป็น TRUE หรือ FALSE ใช้เป็นตัวบอกว่าสถานะของผู้เล่นได้เปลี่ยนไปจากเดิมหรือไม่ ซึ่งจะใช้ในการทำ synchronization ระหว่างผู้เล่นเพื่อป้องกันไม่ให้ผู้เล่นเคลื่อนไหวก่อนที่ผู้เล่นคนอื่นจะรับรู้การเคลื่อนไหวในครั้งก่อนหน้านั้น
- ☐ stat; มีค่าที่เป็นไปได้ในขณะใดขณะหนึ่งคือ DEAD, INIT, READY, ALIVE, WIN, หรือ DYING ซึ่งเป็นตัวบอกสภาวะปัจจุบันของผู้เล่นว่า 'ตาย', 'กำลังจะเริ่มเล่น', 'พร้อมที่จะเริ่มเล่น', 'กำลังเล่น', 'ชนะแล้ว' หรือ 'กำลังจะตาย' ตามลำดับ
- ☐ initT; เป็นตัวเก็บค่าเวลาเมื่อผู้เล่นใหม่ขอ Logon เพื่อจะเริ่มเล่น ใช้ในการตรวจสอบในกรณีที่ผู้เล่นใหม่ขอ Logon แล้วเกิดปัญหาอย่างใดอย่างหนึ่งทำให้ไม่สามารถติดต่อกับมอนิเตอร์อีกได้ โดยมอนิเตอร์จะทำการตรวจสอบว่าผู้เล่นใหม่คนใดที่มีสภาวะ 'กำลังจะเริ่มเล่น' อยู่เป็นเวลาเกินกว่าที่กำหนดไว้ มอนิเตอร์ก็จะจัดการเปลี่ยนสภาวะของผู้เล่นนั้นกลับเป็น 'ตาย' ทันทีเพื่อป้องกันไม่ให้เกิดการเสียจำนวนผู้เล่นที่จะเล่นได้ไปโดยเปล่าประโยชน์
- ☐ lifep; เป็นตัวบอกค่าพลังชีวิตของผู้เล่น ซึ่งมีค่าสูงสุดไม่เกินค่าคงที่ MAXLIFE POWER ที่กำหนดไว้ โดยค่าพลังชีวิตนี้จะลดลงเมื่อผู้เล่นถูกอาวุธ/เวทมนต์ของผู้เล่นคนอื่น และเมื่อลดลงจนถึงศูนย์เมื่อใด ผู้เล่นคนนั้นก็จะเปลี่ยนสภาวะของตนเองเป็น 'กำลังจะตาย' และ ถูกมอนิเตอร์เปลี่ยนสภาวะเป็น 'ตาย' ตามลำดับ ซึ่งมีความหมายเท่ากับว่าผู้เล่นคนนั้นแพ้แล้วนั่นเอง
- ☐ pos; เป็นโครงสร้างข้อมูลแบบ struct ที่ใช้ในการเก็บตำแหน่ง และ ทิศทางปัจจุบันของผู้เล่นในเขาวงกต ซึ่งประกอบไปด้วยฟิลด์ย่อยๆดังนี้
  1. x; ใช้เก็บตำแหน่งของผู้เล่นในแนวดิ่ง (column)
  2. y; ใช้เก็บตำแหน่งของผู้เล่นในแนวนอน (row)
  3. dir; ใช้เก็บทิศทางที่ผู้เล่นหันหน้าไป (direction) โดยมีทิศที่เป็นไปได้อยู่สี่ทิศคือ N, E, S, และ W ตามลำดับ
- ☐ act; ใช้บอกลักษณะท่าทางของผู้เล่นในขณะใดขณะหนึ่ง (action) โดยมีท่าทางที่เป็นไปได้ทั้งหมดคือ STAND, STEPF, STEPB, STEPL, STEPR, TURNL, TURNR, FIRE,

และ HIT ซึ่งหมายถึงการ 'ยืนเฉยๆ', 'ก้าวไปข้างหน้า' , 'ก้าวถอยหลัง', 'ก้าวไปทางซ้าย', 'ก้าวไปทางขวา', 'หันไปทางซ้าย', 'หันไปทางขวา' และ 'ปล่อยอาวุธ/เวทมนต์' ตามลำดับ

- magic; เป็นโครงสร้างข้อมูลแบบ struct ที่ใช้ในการบอก ชนิด ตำแหน่ง ทิศทางการเคลื่อนที่ และข้อมูลอื่นๆ ของอาวุธ/เวทมนต์ ที่ผู้เล่นปล่อยออกไป ซึ่งประกอบไปด้วยฟิลด์ย่อยๆ ดังนี้

  1. pos; ใช้เก็บตำแหน่ง และ ทิศทางการเคลื่อนที่ของอาวุธ/เวทมนต์ มีโครงสร้างเช่นเดียวกับฟิลด์ 'pos' ที่ใช้บอกตำแหน่ง และ ทิศทางของผู้เล่น
  2. distance; ใช้ในการควบคุมระยะทางที่อาวุธ/เวทมนต์จะเคลื่อนที่ไปได้เมื่อถูกปล่อยออกมาแต่ละครั้ง โดยที่ระยะทางสูงสุดจะถูกกำหนดโดยค่าคงที่ MAXFIRE DISTANCE ที่กำหนดไว้ โดยเมื่ออาวุธเคลื่อนที่ไปหนึ่งช่องค่าระยะทางนี้ก็จะถูกลดลงหนึ่ง และเมื่อค่านี้มีค่าเป็นศูนย์ก็จะหมายความว่าไม่ได้มีอาวุธ/เวทมนต์จากผู้เล่นปรากฏอยู่
  3. firep; ใช้ในการบอกพลังในการปล่อยอาวุธ/เวทมนต์ที่เหลืออยู่ของผู้เล่นในขณะใดขณะหนึ่ง โดยค่าของพลังที่ลดลงจากการปล่อยอาวุธ/เวทมนต์แต่ละครั้งนั้นจะแปรผกผันกับชนิดของอาวุธ/เวทมนต์ที่ปล่อยออกไปนั้น โดยถ้ายิ่งเป็นชนิดที่มีอานุภาพร้ายแรงเพียงใด ก็จะทำให้พลังที่ต้องเสียไปจากการปล่อยอาวุธ/เวทมนต์ชนิดนั้นมากขึ้นไปด้วย และถ้าหากว่าพลังนี้มีค่าเป็นศูนย์ก็จะทำให้ผู้เล่นไม่สามารถปล่อยอาวุธได้อีกต่อไป แต่ว่าผู้เล่นจะสามารถเพิ่มพลังนี้ได้โดยการทำการ 'ออกกำลังกาย' ซึ่งก็คือการเดินนั่นเอง โดยที่การเดินหนึ่งก้าวจะได้พลังเพิ่มขึ้นหนึ่งจุด และพลังสูงสุดที่มีได้จะถูกกำหนดโดยค่าคงที่ MAXFIREPOWER ที่ได้กำหนดไว้
  4. fireT; ใช้ในการเก็บเวลาขณะที่ผู้เล่นปล่อยอาวุธ/เวทมนต์ออกไปในแต่ละครั้งเพื่อใช้ประโยชน์ในการควบคุมความเร็วในการเคลื่อนที่ของอาวุธ/เวทมนต์ให้เป็นไปตามค่า คงที่ MAGIC_SPEED ที่กำหนดไว้ โดยความเร็วในการเคลื่อนที่ของอาวุธจะสูงกว่าความเร็วสูงสุดในการเคลื่อนที่ของผู้เล่นที่กำหนดโดยค่าคงที่ MOVE_SPEED ที่กำหนดไว้อยู่ประมาณ 0.1 ถึง 0.2 วินาที

5. type; ใช้ในการบอกชนิดของอาวุธ/เวทมนต์ที่ผู้เล่นปล่อยออกมา ซึ่งจะเป็นตัวกำหนดความร้ายแรงของอาวุธ/เวทมนต์ชนิดนั้นๆ ซึ่งมีชนิดกำหนดไว้แล้วดังนี้คือ ZERO, NORMAL และ อาจกำหนดเพิ่มอีกได้ภายหลัง

- mesg; ใช้ในการส่งข้อความจากผู้เล่นคนหนึ่งไปยังผู้เล่นคนอื่น โดยมีชนิดของข้อความที่กำหนดไว้แล้วดังนี้คือ NONE, HELLO, CURSE, SURRENDER, และ YAHOO โดยถ้าชนิดของข้อความเป็น NONE ก็จะหมายความว่าผู้เล่นคนนั้นไม่มีข้อความใดที่จะส่งไปให้ผู้เล่นอื่นนั่นเอง

3.3.2 แผนที่เขาวงกต

เขาวงกตที่ใช้เล่นจะเก็บไว้ในโครงสร้างข้อมูลแบบอาเรย์สองมิติของจำนวนเต็มชนิด unsigned short โดยขนาดของ อาเรย์ จะเป็นตัวกำหนดขนาดของเขาวงกตซึ่งจะมีขนาดในแนวนอน และแนวตั้งได้ไม่เกินค่าคงที่ MAX_MAZE_SIZE_X และ MAX_MAZE_SIZE_Y ที่กำหนดไว้ตามลำดับ โดยในโปรแกรมตัวอย่างนี้ได้กำหนดค่าทั้งสองไว้ที่ 100 ช่องเท่ากัน

สำหรับการเก็บลักษณะของเขาวงกตแต่ละช่องนั้นจะใช้เก้าบิทด้วยกัน โดยสี่บิทแรกใช้บอกว่ามีประตูอยู่ในทิศใดบ้าง และ สี่บิทถัดไปจะใช้ในการบอกว่ามีกำแพงอยู่ในทิศใดบ้าง โดยจะเรียงทิศแบบวนขวาจากบิทสูงไปหาบิทต่ำ กล่าวคือ บิทสูงสุดจะแทนทิศเหนือ บิทถัดไปจะแทนทิศตะวันออก ทิศใต้ และ ทิศตะวันตกตามลำดับ และ ถ้าทิศใดยังไม่ถูกกำหนดว่ามีกำแพงหรือ ประตู ก็จะถือว่าทิศนั้นมีค่าเป็น undefined ซึ่งจะเป็นสภาวะเริ่มต้นก่อนที่เขาวงกตจะถูกสร้างขึ้นนั่นเอง ส่วนอีกหนึ่งบิทที่เหลือนั้นจะถูกใช้ในระหว่างการสร้างเขาวงกตเท่านั้น

3.3.3 ข้อมูลการแสดงผล

ลักษณะของการแสดงผลของเกมนี้นั้นเป็นกราฟิกบน X Window การแสดงผลส่วนใหญ่เป็นรูปภาพที่คงที่ คือพวกวินโดว์แสดงรูปต่างๆ ปุ่มลูกศรต่างๆ เป็นต้น กับรูปภาพที่มีการเปลี่ยนแปลงตลอดเวลา คือรูปของทางเดินซึ่งจะเปลี่ยนแปลงเมื่อผู้เล่นมีการกดปุ่มเดิน หรือเมื่อมีผู้เล่นคนอื่นเคลื่อนที่เข้ามาในระยะที่เรามองเห็น

โครงสร้างข้อมูลของรูปภาพที่คงที่

โครงสร้างข้อมูลของพวกวินโดว์ต่างๆ จะเป็นตัวแปรที่เป็นชนิดวิดเจต ซึ่งเราต้องมีการสร้างตามลำดับขั้นของวิดเจต ดังนี้

- ☐ Shell เป็นซุปเปอร์คลาสวิดเจตที่สร้างเพื่อรองรับวิดเจตอื่นๆ
- ☐ MainWindow อยู่ภายใต้ Shell อีกทีเป็นวิดเจตวินโดว์หลักเพื่อรองรับวิดเจตอื่นๆ
- ☐ WorkRegion อยู่ภายใต้วิดเจต MainWindow เป็นวิดเจตแบบ BulletinBoard สำหรับเป็นพื้นที่ทำงานของวิดเจตอื่นๆ
- ☐ MenuBar อยู่ภายใต้วิดเจต MainWindow สำหรับสร้างเมนูบาร์
- ☐ PullDown อยู่ภายใต้วิดเจต MenuBar สำหรับสร้างพูลดาวน์เมนู
- ☐ MenuButton เป็นวิดเจตแบบ CascadeButton อยู่ภายใต้วิดเจต PullDown ไว้ใช้สำหรับสร้างปุ่มกดบนเมนูบาร์
- ☐ Label เป็นวิดเจตแบบ PushButton อยู่ภายใต้วิดเจต PullDown ใช้สำหรับเป็นตัวเลือกของพูลดาวน์เมนู
- ☐ MainBoard เป็นวิดเจตแบบ BulletinBoard อยู่ภายใต้วิดเจต WorkRegion มีทั้งหมด 4 อัน ซึ่งก็คือทั้ง 4 ส่วนของหน้าจอ ได้แก่
  1. เป็นวิดเจตรองรับวิดเจตที่วาดรูปทางเดิน
  2. เป็นวิดเจตรองรับวิดเจตที่วาดรูปรายละเอียดของตัวละคร
  3. เป็นวิดเจตรองรับวิดเจตที่แสดงข้อความข่าวสาร ที่เป็นตัวอักษร
  4. เป็นวิดเจตรองรับวิดเจตที่ใช้สำหรับสร้างปุ่มลูกศร
- ☐ DrawingArea เป็นวิดเจตแบบ DrawingArea อยู่ภายใต้วิดเจต MainBoard ใช้สำหรับการวาดรูป
- ☐ Label เป็นวิดเจตแบบ Label อยู่ภายใต้วิดเจต MainBoard ใช้สำหรับการแสดงรูปทางเดิน
- ☐ MessageBoard เป็นวิดเจตแบบ Label อยู่ภายใต้วิดเจต MainBoard ใช้สำหรับแสดงข้อความข่าวสารที่เป็นตัวอักษร
- ☐ InfoLabel เป็นวิดเจตแบบ Label อยู่ภายใต้วิดเจต MainBoard ใช้สำหรับแสดงรูปรายละเอียดของตัวละคร

- PadArea เป็นวิดเจตแบบ Form อยู่ภายใต้วิดเจต MainBoard ใช้สำหรับรองรับวิดเจตที่ใช้สร้างปุ่มกด
- ArrowBtn เป็นวิดเจตแบบ PushButton อยู่ภายใต้วิดเจต PadArea ใช้สำหรับทำเป็นปุ่มลูกศร

<u>โครงสร้างข้อมูลของรูปภาพที่มีการเปลี่ยนแปลง</u>

สำหรับโครงสร้างข้อมูลของรูปภาพที่มีการเปลี่ยนแปลงนั้น จะเก็บไว้เป็น Struct โดยมีฟิลด์ 6 ฟิลด์ดังนี้

1. id คือหมายเลขประจำตัวของรูปภาพๆ นั้น
2. pixmap ใช้เก็บข้อมูลรูปภาพตามแบบโครงสร้างข้อมูลแบบ Pixmap
3. img_raster ใช้เป็นที่เก็บข้อมูลดิบของรูปที่ Load เข้ามา
4. img ใช้เก็บข้อมูลรูปภาพตามแบบโครงสร้างข้อมูลแบบ XImage
5. x, y เก็บตำแหน่งของรูปๆ นั้น
6. width, height เก็บความกว้างและความสูงของรูปๆ นั้น

สำหรับรูปที่จะ Load เข้ามานั้นมีรูปแบบเป็นไฟล์แบบ GIF (.GIF Format) โดยมีฟังก์ชันสำเร็จรูปในการแปลงภาพจาก GIF ไฟล์ ไปเป็นข้อมูลแบบ XImage หลังจากนั้นก็ต้องใช้คำสั่ง XCreatePixmap สำหรับจองเนื้อที่ในหน่วยความจำให้เป็นแบบพิกแมป (พิกแมปเป็นหน่วยความจำที่ใช้สำหรับเก็บข้อมูลที่จะใช้แสดงผล แต่จะมองไม่เห็นเนื่องจากเป็นที่เก็บเฉยๆ ถ้าต้องการให้เห็นจะต้องคัดลอกไปไว้ในวินโดว์) เมื่อได้ข้อมูลแบบ XImage กับ พิกแมปมาแล้วก็ใช้คำสั่ง XPutImage เพื่อนำข้อมูลจาก XImage เข้าไปใส่ในพิกแมป หลังจากนั้นก็นำพิกแมปที่ได้นี้ไปวาดทับบนวิดเจต Label

สำหรับรูปของทางเดินนั้นจะเป็นพิกแมปที่เก็บรูปเฉพาะส่วนไว้ คือภายในทางเดินนั้นจะมีการแบ่งออกเป็นบล็อกๆ 1 บล็อกเท่ากับ 1 ช่องในเขาวงกต และผู้เล่นสามารถมองเห็นไปข้างหน้าได้อีก 4 บล็อก ดังนั้นรูปที่เราจะเห็นจะแบ่งออกเป็น 11 ส่วนคือ

1. รูปทางด้านซ้ายและขวา ของบล็อกที่ผู้เล่นยืนอยู่
2. รูปทางด้านซ้ายและขวา ของบล็อกข้างหน้า 1 บล็อกถัดจากที่ผู้เล่นยืนอยู่

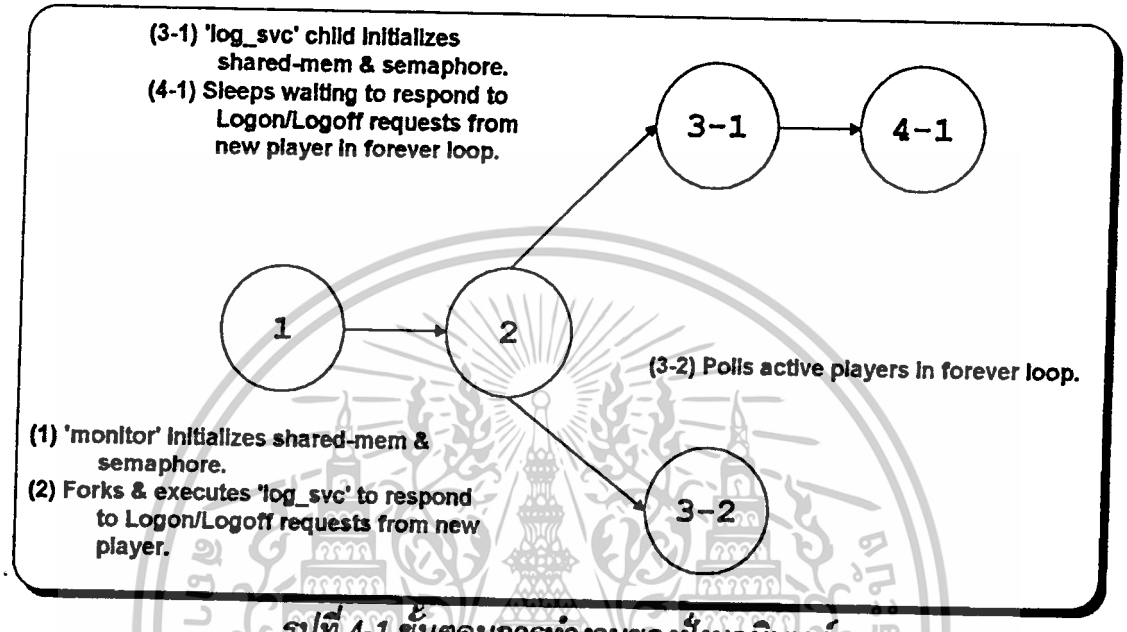
3. รูปทางด้านซ้ายและขวา ของบล็อกข้างหน้า 2 บล็อกถัดจากที่ผู้เล่นยืนอยู่
4. รูปทางด้านซ้ายและขวา ของบล็อกข้างหน้า 3 บล็อกถัดจากที่ผู้เล่นยืนอยู่
5. รูปทางด้านซ้ายและขวา ของบล็อกข้างหน้า 4 บล็อกถัดจากที่ผู้เล่นยืนอยู่
6. รูปกำแพงหรือทางเดินที่อยู่ถัดจาก 4 บล็อกข้างหน้าไป

ซึ่งรูปที่จะมาแสดงนี้อาจจะไม่ใช่ทั้ง 11 รูปก็ได้ อาจจะเป็นแค่ 9 รูป คือในกรณีที่บล็อกที่ 4 เป็นกำแพง ดังนั้นจะมองเห็นไปข้างหน้าได้เพียง 3 บล็อกเท่านั้น หรือ 7, 5 และ 3 รูปก็ได้ ถ้ามองเห็นไปข้างหน้าได้เพียง 2, 1 และมีกำแพงอยู่ตรงหน้า และสำหรับรูปด้านข้างซ้ายและขวานั้น มีโอกาสเป็นไปได้อีก 3 กรณีคือ

1. ด้านข้างนั้นเป็นกำแพง
2. ด้านข้างนั้นเป็นช่องว่างและเป็นทางตัดกัน
3. ด้านข้างนั้นเป็นช่องว่างและทางเดินนั้นเป็นทางเดินขนานกัน

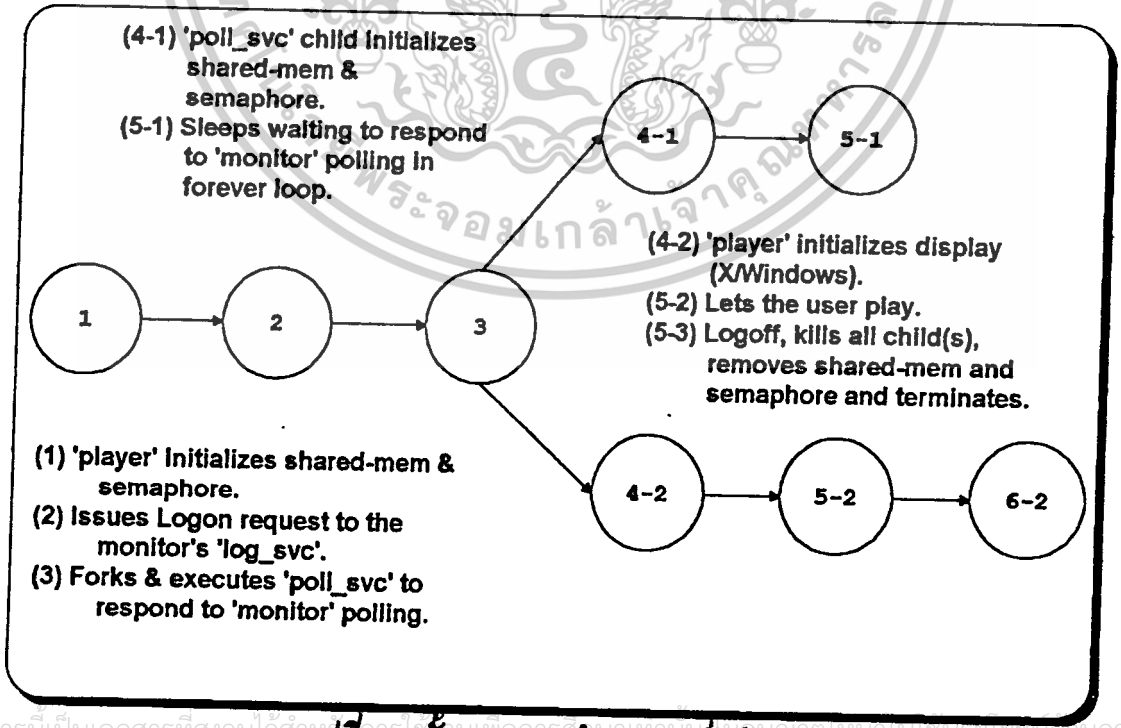
บทที่ 4
อัลกอริทึม

☐ สำหรับขั้นตอนการทำงานของฝั่งมอนิเตอร์นั้นเป็นไปดังรูปที่ 4-1



รูปที่ 4-1 ขั้นตอนการทำงานของฝั่งมอนิเตอร์

☐ ส่วนขั้นตอนการทำงานของฝั่งผู้เล่นนั้นเป็นไปดังรูปที่ 4-2



รูปที่ 4-2 ขั้นตอนการทำงานของฝั่งผู้เล่น

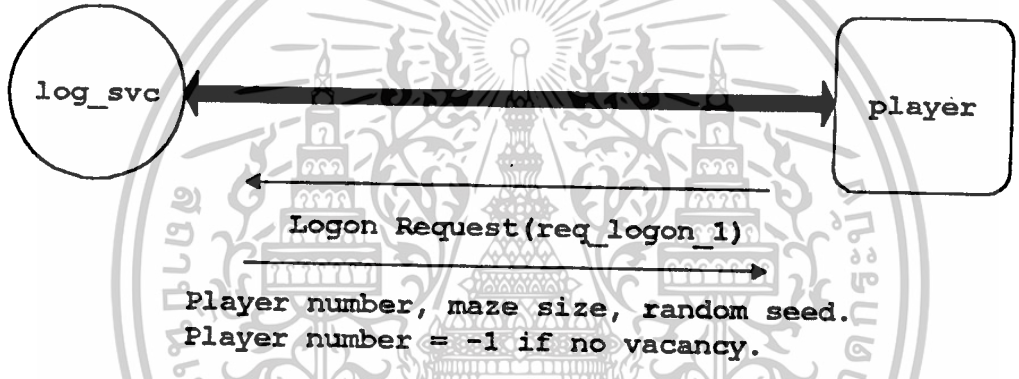
อัลกอริทึมการทำงานในส่วนต่างๆโดยละเอียด แบ่งตามโครงสร้างจากรูปที่ 3-1

4.1 Networking

การติดต่อระหว่างผู้เล่น (Player) กับ มอนิเตอร์ (Monitor) แต่ละคู่จะประกอบไปด้วยสอง connections ด้วยกัน ซึ่งได้แก่

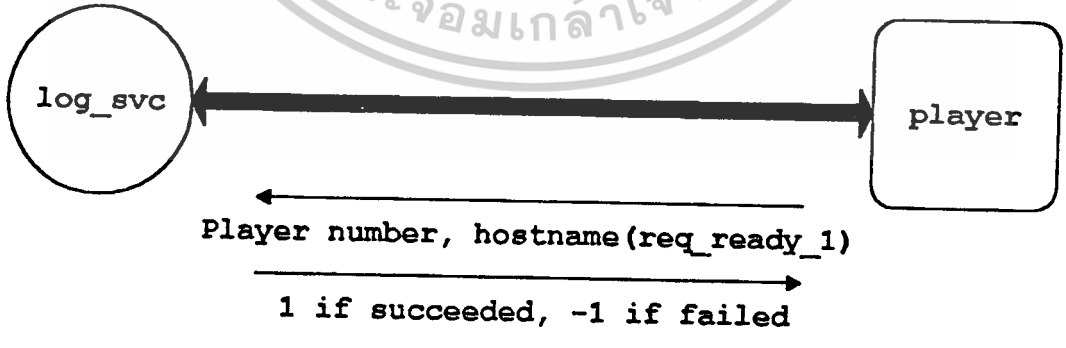
- ☐ การติดต่อระหว่างโปรเซส 'player' ที่ฝั่งผู้เล่น กับ โปรเซส 'log_svc' ที่ฝั่งมอนิเตอร์ ซึ่งจะเริ่มขึ้นเมื่อผู้เล่นต้องการขอ Logon และ จะสิ้นสุดลงเมื่อผู้เล่นขอ Logoff หรือ เมื่อผู้เล่นอยู่ในสถานะ 'กำลังจะตาย' โดยจะมีการติดต่อเป็นขั้นๆ ดังต่อไปนี้

► สภาวะของผู้เล่น: 'ตาย' -> 'กำลังจะเริ่มเล่น'



รูปที่ 4.1-1 ผู้เล่นขอเริ่มเล่น

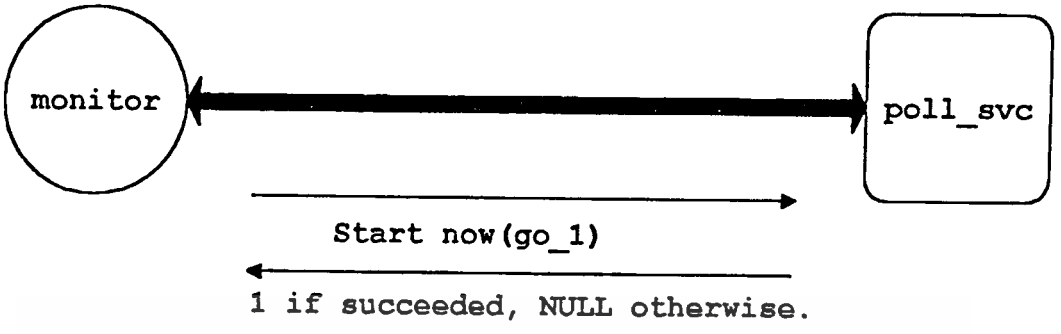
► สภาวะของผู้เล่น: 'กำลังจะเริ่มเล่น' -> 'พร้อมที่จะเริ่มเล่น'



รูปที่ 4.1-2 ผู้เล่นพร้อมที่จะเริ่มเล่น

- ☐ การติดต่อระหว่างโปรเซส 'monitor' ที่ฝั่งมอนิเตอร์ กับ โปรเซส 'poll_svc' ที่ฝั่งผู้เล่น ซึ่งจะเริ่มขึ้นเมื่อผู้เล่นอยู่ในสถานะ 'พร้อมที่จะเริ่มเล่น' และ จะสิ้นสุดลงเมื่อผู้เล่นอยู่ในสถานะ 'ตาย'

► สภาวะของผู้เล่น: 'พร้อมที่จะเริ่มเล่น' -> 'กำลังเล่น

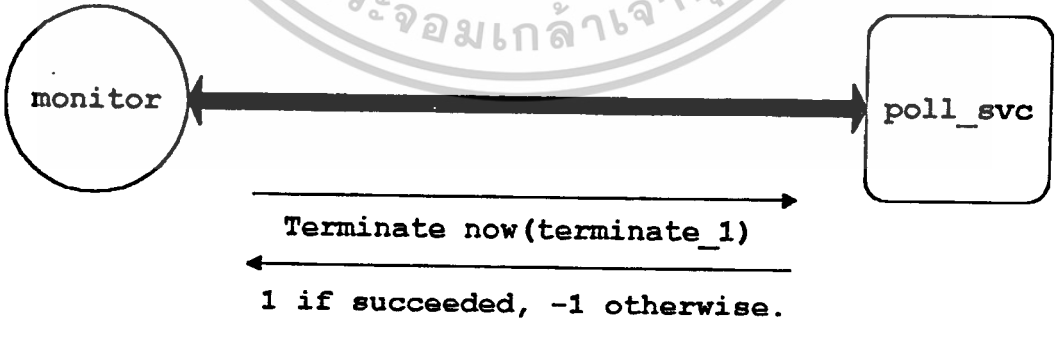


รูปที่ 4.1-3 ให้เริ่มเล่นได้

► สภาวะของผู้เล่น: 'กำลังเล่น'

รูปที่ 4.1-4 ขณะกำลังเล่น

► สภาวะของผู้เล่น: 'กำลังตาย' -> 'ตาย'



รูปที่ 4.1-5 สิ้นสุดการเล่น

4.2 Processing

☐ การสร้างเขาวงกต

สำหรับส่วนของโปรแกรมที่ใช้ในการสร้างเขาวงกตนั้นผู้จัดทำได้มาจาก Internet โดยมีขั้นตอนการทำงานคล้ายกับการ depth-first traverse ไปใน tree ดังนี้

1. เริ่มแรกนั้นทุกๆช่องในเขาวงกตจะอยู่ในสภาพ undefined คือ ไม่มีทั้งกำแพง และประตู
2. จัดการเลือกจุดเริ่มต้น ณ ที่ใดที่หนึ่งที่อยู่ริมขอบโดยการสุ่ม
3. ตรวจแต่ละด้านของช่องปัจจุบัน ถ้าหากด้านใดยังอยู่ในสภาพ undefined และด้านตรงข้ามของช่องที่อยู่ติดกับด้านนั้นเป็นกำแพง ด้านนั้นก็จะต้องเป็นกำแพงด้วย
4. เลือกด้านที่ยังเป็น undefined จากช่องปัจจุบันมาด้านหนึ่งโดยการสุ่ม เปลี่ยนให้เป็นประตู แล้วเปลี่ยนไปอยู่ในช่องถัดไป
5. ทำตามข้อ (3) และ (4) ไปจนกว่าจะไม่มีช่องที่ยังมีด้านที่มีสภาวะเป็น undefined เหลืออยู่

☐ ระหว่างการเล่น

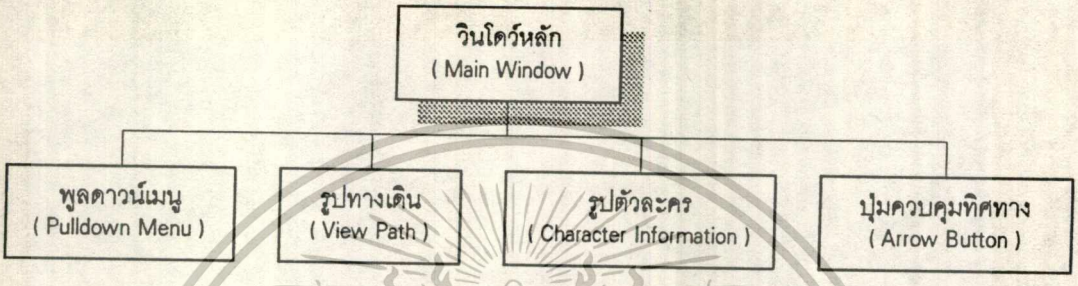
► การเคลื่อนที่/เปลี่ยนทิศทาง; ผู้เล่นสามารถเคลื่อนที่ครั้งละหนึ่งช่องได้ทั้งสี่ทิศทางคือ ไปข้างหน้า ถอยหลัง ไปทางซ้าย และ ไปทางขวา โดยจะมีโมดูลที่ทำหน้าที่ตรวจสอบว่าผู้เล่นสามารถเคลื่อนที่ไปได้หรือไม่ เช่น ตรงหน้านั้นเป็นกำแพง หรือ เป็นประตู ที่ช่องถัดไปนั้นมีผู้เล่นอื่นยืนอยู่ก่อนแล้วหรือไม่ เป็นต้น ส่วนการหมุนตัวเปลี่ยนทิศทางนั้น จะสามารถหมุนไปทางซ้ายหรือขวาครั้งละเก้าสิบองศาเท่านั้น โดยที่การหมุนตัวนั้น สามารถทำได้ตลอดเวลา ไม่ว่าจะอยู่ ณ ตำแหน่งใดในเขาวงกต

► การต่อสู้; เมื่อผู้เล่นสองคนเดินมาพบกันเข้าจะสามารถต่อสู้กันได้ โดยการปล่อยอาวุธเข้าใส่ฝ่ายตรงข้าม ซึ่งก็จะต้องมีการตรวจสอบว่าสามารถปล่อยเวทมนต์ได้หรือไม่ เช่น ถ้าหากตรงหน้านั้นเป็นกำแพง ก็จะไม่สามารถปล่อยเวทมนต์ออกไปได้ เพราะว่ากำแพงของเขาวงกตนั้นสามารถป้องกันอาวุธได้ เป็นต้น และ ถ้าปล่อยเวทมนต์ออกไปแล้วก็จะต้องตรวจสอบอีกว่าเวทมนต์นั้นเคลื่อนที่ไปถูกผู้เล่นอื่นหรือกำแพงหรือไม่ และ ในขณะเดียวกันก็จะต้องตรวจสอบว่าตนเองนั้นได้ถูกอาวุธของผู้เล่นคนอื่นเข้าหรือไม่ แล้วจึงจัดการ

เปลี่ยนแปลงสถานะของผู้เล่นให้ถูกต้อง เช่น เมื่อปล่อยอาวุธออกไปพลังในการปล่อยอาวุธก็จะต้องลดลง และ ถ้าถูกอาวุธของผู้อื่นพลังชีวิตก็จะต้องลดลง เป็นต้น

4.3 User Interface

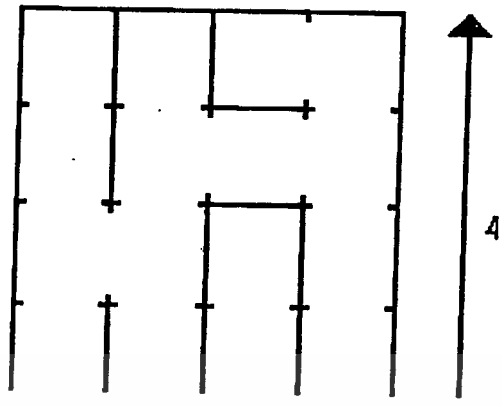
รูปแบบของหน้าจอที่จะติดต่อกับผู้เล่นนั้นมีการออกแบบไว้ดังนี้คือ จะเป็นวินโดว์ที่มีขนาดคงที่ มีการแบ่งหน้าจอออกเป็นส่วนๆ 4 ส่วนคือ ส่วนรูปทางเดิน (View Path) ส่วนรูปแสดงสถานะของผู้เล่น ส่วนรูปแสดงข้อความต่างๆ ส่วนปุ่มลูกศรแสดงทิศทาง สำหรับการโปรแกรมนั้นได้แบ่งส่วนของโปรแกรมที่สำคัญๆ ออกเป็นส่วนๆ ดังนี้



รูปที่ 4.3-1 โครงสร้างของรูปแบบหน้าจอ

สำหรับวินโดว์หลักนั้นจะกำหนดขนาดให้คงที่ไว้แน่นอน เพื่อให้ง่ายต่อการเขียนโปรแกรมและมีรูปแบบและสัดส่วนที่สวยงาม ในส่วนของเมนูแบบพูลดาวน์นั้นจะมี Push Button ที่เป็นคำสั่งต่างๆ เช่น Start Game หรือ Quit เป็นต้น และในส่วนของวินโดว์ที่วาดรูปทางเดินนั้น พื้นที่วาดรูปภายในจะแบ่งไว้เป็น 11 ส่วนดังรูปที่ 4.3-2 โดยแต่ละส่วนจะเป็นบล็อก (Block) ของเขาวงกต โดยผู้เล่นจะมองเห็นไปข้างหน้าเป็นจำนวน 4 ช่องตามรูปที่ 4.3-3

รูปที่ 4.3-2 ภาพแสดงทางเดินในเขาวงกต



รูปที่ 4.3-3 แสดงภาพทางเดินและจำนวนบล็อกที่สามารถมองเห็นได้

การควบคุมการแสดงรูปตามลักษณะทางเดินของเขาวงกตจะใช้วิธีการแบ่งรูปออกเป็น 11 ส่วน แต่ละส่วนจะถูกคำนวณว่าอยู่ในตำแหน่งที่ถูกต้องตามทางเดิน เช่นถ้าด้านขวามือของตำแหน่งที่อยู่ขณะนั้นมีกำแพง ก็จะนำภาพกำแพงมาวาดทับ และตรวจดูด้านซ้ายมือว่ามีกำแพงหรือไม่ ถ้ามีก็นำภาพกำแพงมาวาดทับเช่นเดียวกัน แล้วก็ตรวจดูช่องถัดไปข้างหน้าว่ามีกำแพงหรือไม่ ถ้าไม่มีก็จะดูทางซ้ายมือของช่องข้างหน้าว่ามีกำแพงหรือไม่ ทำอย่างนี้จนครบ 5 ครั้ง (ตำแหน่งที่เรายืนอยู่นับเป็น 1 ช่องบวกกับตำแหน่งข้างหน้าอีก 4 ช่อง) แต่ถ้าหากว่าตำแหน่งด้านหน้าของเราตำแหน่งไหนเป็นทางตัน คือมีกำแพง ก็จะนำรูปกำแพงของตำแหน่งนั้นมาวาดทับลงไป และเมื่อมีการเดินก็จะต้องตรวจดูก่อนว่าเดินได้หรือไม่ ถ้าได้ก็จะทำการคำนวณรูปนำมาวาดทับเหมือนเดิม และเวลาที่พบผู้เล่นคนอื่นก็จะนำรูปของผู้เล่นคนนั้นมาวาดทับไปบนรูปทางเดิน โดยจะวาดทับไปในตำแหน่งที่ผู้เล่นคนนั้นอยู่ สำหรับรูปที่จะนำมาวาดทับนี้จะถูกเก็บอยู่ในพิกแมปในหน่วยความจำทำให้มีความเร็วในการวาดทับรวดเร็วมากพอสมควร

สำหรับวินโดว์ที่แสดงรูปของตัวละครนั้นเป็นวิดเจต Drawing Area ธรรมดา ที่มีการนำเอารูปภาพที่เตรียมไว้ขึ้นมาแสดงเฉยๆ และสำหรับวินโดว์แสดงข้อความก็จะเป็นวินโดว์ที่ใช้สำหรับแสดงข้อความตอบโต้กับผู้เล่นคนอื่นแต่ละคน และสำหรับส่วนที่เป็นปุ่มควบคุมทิศทางนั้น จะเป็นปุ่มที่มีรูปภาพของลูกศรแบบต่างๆ 6 แบบวาดทับอยู่ (ลูกศรชี้ไปทางซ้าย ,ชี้ไปทางขวา, ชี้ขึ้นข้างบนเป็นต้น โดยแต่ละอันมีหน้าที่ดังนี้

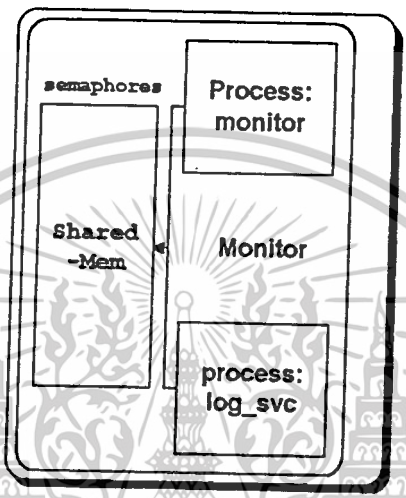
- ☐ ลูกศรชี้หันไปทางซ้ายและขวา คือการหันตัวผู้เล่นไปทางซ้ายและทางขวา 90°
- ☐ ลูกศรชี้ขึ้นและลง คือการเคลื่อนที่ไปขางหน้าและถอยหลัง 1 บล็อก

- ลูกศรชี้ซ้ายและขวา คือการเลื่อนตัวผู้เล่นไปทางซ้ายและทางขวา 1 บล็อกโดยไม่ได้หันหน้าไป

4.4 Synchronization Method

สำหรับวิธีที่ใช้ในการควบคุมความสัมพันธ์กันของการเปลี่ยนแปลงข้อมูลสถานะของผู้เล่นในเขาวงกตให้เป็นไปอย่างสอดคล้อง (synchronously) กันนั้น อาจแบ่งออกได้เป็นสองส่วนคือ

4.4.1 การควบคุมระหว่างโปรเซส



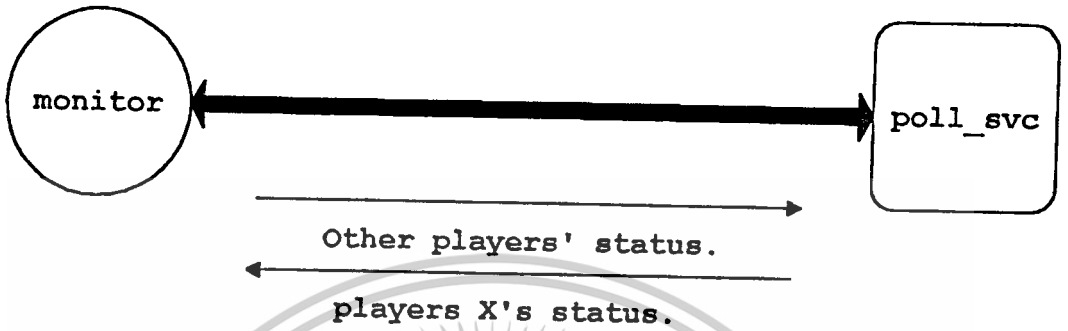
รูปที่ 4.4-1 การควบคุมระหว่างโปรเซส

ส่วนที่ใช้ควบคุมการเปลี่ยนแปลงข้อมูลสถานะผู้เล่นระหว่างโปรเซส ซึ่งได้แก่ การเปลี่ยนแปลงสภาวะผู้เล่นโดยโปรเซส 'logon_svc' กับโปรเซส 'monitor' ที่ฝั่งมอนิเตอร์ และการเปลี่ยนแปลงข้อมูลสถานะผู้เล่น และ การรับสถานะของผู้เล่นอื่นของโปรเซส 'poll_svc' กับโปรเซส 'player' ที่ฝั่งผู้เล่นนั่นเอง ซึ่งได้เลือกใช้การทำหน่วยความจำร่วมกันโดยการใช้ตัวแปรที่เก็บสถานะของผู้เล่นแต่ละคนไว้ร่วมกันระหว่างโปรเซส และ ใช้เซมาฟอร์ในการควบคุมการเข้าถึง (access) และ การแก้ไข (modify) โดยการใส่ wait และ signal ไว้ก่อนและหลังส่วนที่มีการเข้าถึงและ/หรือ แก้ไขข้อมูลสถานะของผู้เล่น ทั้งระหว่างโปรเซส 'logon_svc' กับโปรเซส 'monitor' ที่ฝั่งมอนิเตอร์ และ ระหว่างโปรเซส 'player' กับโปรเซส 'poll_svc' ที่ฝั่งผู้เล่น

4.4.2 การควบคุมระหว่างผู้เล่น

ส่วนที่ใช้ควบคุมความสอดคล้อง และ ความถูกต้องตรงกัน (consistency) ของสถานะผู้เล่น ณ ทุกๆโหนด (node) ในขณะใดขณะหนึ่ง เช่น ถ้าหากว่าในข้อมูลสถานะบ่งว่ามีผู้เล่นอื่นยืนอยู่ตรงหน้า และ/หรือ มีอาวุธกำลังเคลื่อนที่เข้ามาจากทิศตรงหน้าแล้ว บนจอภาพก็จะต้องแสดงรูปลักษณะท่าทางของผู้เล่น และ/หรือ อาวุธนั้นให้เห็นอย่างถูกต้อง และ ถ้าหากว่ามี

การปล่อยอาวุธจากผู้เล่นคนใดคนหนึ่ง หรือ หลายคนพร้อมๆกันก็ตาม จะต้องสามารถตรวจสอบได้ว่าอาวุธนั้นได้ไปถูกผู้เล่นคนใดหรือไม่ และ ทำการลดพลังการใช้อาวุธ และ/หรือ พลังชีวิตได้อย่างถูกต้องตามชนิดของอาวุธนั้นๆ โดยได้เลือกใช้หลายวิธีร่วมกันดังนี้



รูปที่ 4.4-2 การควบคุมระหว่างผู้เล่น

1. ใช้แฟลก 'changed' ในโครงสร้างข้อมูลสถานะของผู้เล่นแต่ละคนเป็นตัวควบคุมไม่ให้ผู้เล่นทำการเปลี่ยนแปลงสถานะของตนเองก่อนที่การเปลี่ยนแปลงสถานะครั้งก่อนหน้านั้นได้ถูกส่งไปให้ผู้เล่นอื่นแล้ว โดยเมื่อใดผู้เล่นมีการเปลี่ยนแปลงสถานะของตนในโปรเซส 'player' 'changed' ก็จะถูกเปลี่ยนค่าเป็น TRUE และจะคงอยู่อย่างนั้นจนกระทั่งถูกโปรเซส 'poll_svc' เปลี่ยนกลับเป็น FALSE หลังจากที่ได้ส่งสถานะใหม่นั้นกลับไปให้มอนิเตอร์เท่านั้น

2. การควบคุมความเร็วในการเคลื่อนที่ของผู้เล่น และ อาวุธที่ปล่อยออกไป โดยนอกจากจะทำการตรวจสอบแฟลก 'changed' ว่าสถานะเดิมได้ถูกส่งไปให้ผู้เล่นอื่นแล้วยังมีการหน่วงเวลาต่ำสุดระหว่างการเดินแต่ละก้าว และ การปล่อยอาวุธแต่ละครั้งอีกด้วย เพื่อเป็นการรับประกันได้ว่าในระหว่างเวลาที่หน่วงอยู่นั้นข้อมูลสถานะของผู้เล่นได้ถูกส่งต่อไปให้ผู้เล่นคนอื่นๆครบถ้วนเรียบร้อยแล้ว นอกจากนี้ยังมีการควบคุมความเร็วในการเคลื่อนที่ของอาวุธที่ปล่อยออกไปอีกด้วย โดยการหน่วงเวลาระหว่างการเดินแต่ละก้าว และ การปล่อยอาวุธแต่ละครั้งนั้น จะกระทำในโปรเซส 'player' ส่วนการควบคุมความเร็วในการเคลื่อนที่ของอาวุธที่ถูกปล่อยออกไปนั้น จะกระทำในโปรเซส 'poll_svc' ทุกครั้งที่ถูก poll จากมอนิเตอร์

3. การตรวจสอบตำแหน่งของผู้เล่นอื่น และตำแหน่งของอาวุธที่ถูกปล่อยออกมาเทียบกับตำแหน่งปัจจุบันของตนเอง ซึ่งใช้ในการตรวจสอบว่าผู้เล่นสามารถเคลื่อนที่ หรือปล่อยอาวุธไปในทิศทางที่ต้องการได้หรือไม่ และ การตรวจสอบว่าผู้เล่นได้ถูกอาวุธ

ของผู้เล่นอื่นหรือไม่ โดยการตรวจสอบการเคลื่อนที่นั้น จะกระทำในโปรเซส 'player' ส่วนการตรวจสอบการถูกอาวุธนั้น จะกระทำในโปรเซส 'poll_svc' ทุกครั้งที่ถูก poll จากมอนิเตอร์

บทที่ 5
สรุปและวิจารณ์

สรุป

โครงงานนี้ได้ทำการพัฒนาโปรแกรมเกมแบบเล่นหลายคนบนเครือข่าย โดยในระหว่างการเล่นผู้เล่นแต่ละคนจะเห็นรูปทางเดินเป็นกราฟิก 3 มิติ การเล่นจะต้องมีการแลกเปลี่ยนข้อมูลสถานะปัจจุบันกับผู้เล่นคนอื่นโดยมีมอนิเตอร์ทำหน้าที่ในการควบคุมการรับ-ส่ง ข้อมูล สำหรับผลของการควบคุมความสอดคล้องและการควบคุมความถูกต้องตรงกันของข้อมูลนั้นสามารถมองเห็นได้อย่างชัดเจนจากการแสดงผลในระหว่างการเล่น

ปัญหาที่พบ

- ☐ โครงงานที่เกี่ยวกับเรื่องการประมวลผลแบบกระจายในประเทศไทยหาได้ยากมาก จึงต้องพึ่งข้อมูลจากต่างประเทศ ซึ่งส่วนมากมักเป็นโครงงานในระดับมหาบัณฑิตขึ้นไป
- ☐ หนังสืออ้างอิงของระบบการประมวลผลแบบกระจายในประเทศไทยหาได้ยากและมีราคาแพงมาก
- ☐ ระบบเครือข่ายของศูนย์ CAD/CAM นั้นต้องรองรับการใช้งานโปรแกรมประเภท CAD/CAM เช่น Autocad และ Mentor Graphics อยู่เป็นประจำ ซึ่งจำเป็นต้องอาศัยการส่งผ่านข้อมูลผ่านเครือข่ายเป็นปริมาณมหาศาลในระหว่างการทำงานเนื่องจากการใช้ NFS และ Clustering ของระบบ ทำให้ไม่สามารถการทดสอบโปรแกรมที่สภาวะใกล้เคียงอุดมคติได้
- ☐ ตัวอย่างแนวทางพัฒนาระบบ X Window นั้นมีน้อย และไม่หลากหลาย

ข้อเสนอแนะและแนวทางการพัฒนาต่อ

- ☐ ส่วนการแสดงผลด้วย X Windows นั้นยังไม่ได้พัฒนาให้เสร็จสมบูรณ์ตามโครงสร้างข้อมูลที่วางไว้ ซึ่งได้แก่ ส่วนของวินโดว์ที่ใช้แสดงสถานะปัจจุบันของผู้เล่น ส่วนที่ใช้แสดง message ต่างๆในระหว่างการเล่น เช่น error message และ message จากผู้เล่นอื่นๆ เป็นต้น และ ส่วนที่ใช้แสดงแผนที่ขนาดเล็ก หรือ ไฟเตือนให้ผู้เล่นสามารถทราบได้ว่ามีผู้เล่นอื่นอยู่ในบริเวณใกล้เคียงกันหรือไม่

- ยังไม่ได้ทำการทดลองการเล่นในสภาพแวดล้อมที่มีผู้เล่นมากกว่า 4 คน และการเล่นผ่านเครื่องคนละประเภท การพัฒนาต่ออาจจะทำในส่วนนี้เพื่อทำให้โปรแกรมมีความคล่องตัวในการใช้งาน และต้องมีการจัดการความสอดคล้องและควบคุมความถูกต้องของข้อมูลให้เหมาะสมกับจำนวนผู้เล่นที่มากขึ้น
- สำหรับรูปแบบของเกมนั้สามารถนำไปพัฒนาต่อเพื่อให้เป็นแบบ Role Play Game (RPG) ได้ โดยการเพิ่มสัตว์ประหลาดที่อาจควบคุมโดยมอนิเตอร์ หรือโปรเซสของผู้เล่นเองก็ได้ อาจมีการเพิ่มระดับขั้นของเขาวงกตเมื่อผู้เล่นสามารถหาทางออกจากเขาวงกตเดิมได้ต่อไป และ อาจมีการสร้างเนื่อเรื่องขึ้น และ มีปริศนาต่หรือ ของต่าง ๆ ที่จะต้องแก้หรือตามหาเพื่อให้ได้คะแนนหรือพลังเพิ่มขึ้น เป็นต้น

บทวิจารณ์

ระบบการประมวลผลแบบกระจายจำเป็นต้องใช้แนวการคิดและการออกแบบที่ไม่เหมือนการพัฒนาโปรแกรมธรรมดาทั่วๆ ไป มีปัญหาต่างๆ ที่จะต้องพิจารนาค่อนข้างมาก และรูปแบบของโครงงานที่เป็นเกมเป็นสิ่งที่น่าสนใจ และท้าทาย และนอกจากโครงงานจะออกมาตรงตามที่ได้ตั้งเป้าหมายไว้แล้ว แต่ก็ยังสามารถพัฒนาต่อเพื่อนำไปแสดงในงานลาดกระบังนิทรรศน์ครั้งต่อไปได้

สำหรับการทำการส่งข้อมูลโดยใช้ RPC นั้นอาจจะประยุกต์ไปใช้กับงานด้านอื่น เช่นการทำ Electronic Data Interchange (EDI) หรือ ระบบการประมวลผลแบบ Client/Server เป็นต้น

ภาคผนวก

สำหรับ source code listing และหนังสือคู่มือการใช้งานโปรแกรมนั้น สามารถขอทำสำเนาได้ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

กิตติกรรมประกาศ

ผู้จัดทำขอขอบพระคุณบุคคลต่างๆที่ได้ให้ความช่วยเหลือในการทำการวิจัยจนสำเร็จลุล่วงไปด้วยดีดังนี้

- คุณพ่อ คุณแม่ พี่น้อง และ ผู้ที่ช่วยเป็นกำลังใจ และ ให้การสนับสนุนในการศึกษาตลอดมา
- อาจารย์ ดร. บุญธีร์ เครือตราชู และอาจารย์ อภิเนตร ฤนากูล อาจารย์ที่ปรึกษาทั้งสองท่านที่ได้ให้คำปรึกษา, คำแนะนำ และข้อมูลที่มีประโยชน์ในการดำเนินงานมาโดยตลอด
- อาจารย์วัชระ ฉัตรวิริยะ ที่ได้ให้คำแนะนำและข้อมูลต่างๆในระหว่างการหาข้อมูล
- พี่ณรงค์ศักดิ์ (พี่เจี๊ยบ) ผู้ควบคุมระบบเครือข่ายของศูนย์ CAD/CAM พี่นภัทร สระเอี่ยม (พี่ปู) ผู้ควบคุมระบบเครือข่ายของภาควิชาฯ และอาจารย์ทุกๆท่านที่ได้ให้ความช่วยเหลือ และอำนวยความสะดวกในการใช้เครื่อง
- เพื่อนๆ พี่ๆ น้องๆ ที่ได้ช่วยเหลือ และให้คำแนะนำในการทำงาน
- บุคคลทุกๆท่านที่ทำให้ Internet เกิดขึ้นและดำรงอยู่เป็นแหล่งข้อมูลทางการศึกษาที่ยิ่งใหญ่ที่สุดในโลก

เอกสารอ้างอิง

1. Adrew Oram and Steve Talbatt, "Managing Projects with Make", O'Reilly & Associates, Inc., 149 p, 1993
2. Adrian Nye, "Volume 0, X Protocol Reference Manual for X Version 11", O'Reilly & Associates, Inc., 414 p., 1989
3. Adrian Nye, "Volume 1, Xlib Programming Manual for Version 11", O'Reilly & Associates, Inc., 659 p., 1989
4. Adrian Nye, Volume 2, Xlib Reference Manual for Version 11", O'Reilly & Associates, Inc., 723 p., 1989
5. Adrian Nye and Tim O'Reily, "Volume 4, X Toolkit Intrinsics Programming Manual OSF/Motif 1.2 Edition", O'Reilly & Associates, Inc., 706 p., 1992
6. David Flanagan, "Volume 5, X Toolkit Intrinsics Reference Manual for X11 Release 4 and Release 5", O'Reilly & Associates, Inc., 916 p., 1992
7. Donald L. McMinds, "Mastering OSF/Motif™ Widgets", Addison - Wesley Publishing Company, Inc., 660 p., 1992
8. John Bloomer, "Power Programming with RPC", O'Reilly & Associates, Inc., 486 p, 1992
9. "Programming with Xlib", HEWLETT PACKARD Company , 1991
10. Steven Mikes, "X Window System™ Technical Reference", Addison - Wesley Publishing Company, Inc., 786 p., 1990
11. Susann Ragsdale, "Parallel Programming", Mc-Graw Hill, 150p, 1992
12. W. Richard Stevens, "UNIX Network Programming", Prentice - Hall, Inc., 772 p., 1990
13. "X Window System Programming : Xlib Student Workbook", HEWLETT PACKARD Company, 1989