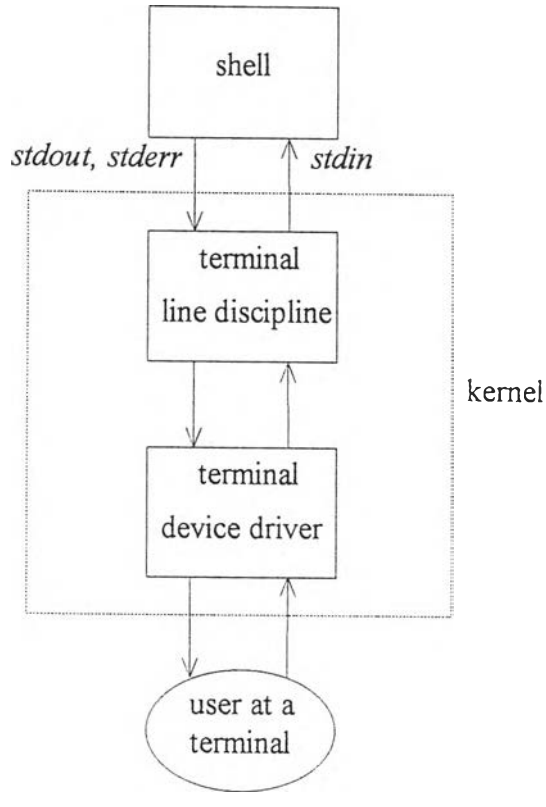
บทที่ 2

ทฤษฎี และความรู้พื้นฐาน

ในบทนี้จะได้กล่าวถึงทฤษฎี และความรู้พื้นฐานที่สำคัญในการทำวิทยานิพนธ์ ซึ่งจะทำให้เข้าใจถึงงานวิจัยได้รวดเร็ว และในตอนท้ายของบทจะเป็นการให้ความหมายของคำศัพท์ที่ใช้เพื่อให้มีความเข้าใจที่ตรงกัน

เทอร์มินัลเทียม (pseudo-terminals)

โปรแกรมขับเทอร์มินัลมีส่วนประกอบที่สำคัญคือ ไลน์ดิสซิพรินส์ (line desciplines) ซึ่งทำให้โปรแกรมขับเทอร์มินัลมีความซับซ้อนมาก เทอร์มินัลถูกสมมติให้เป็นเสมือนอุปกรณ์สื่อสารสองทางเต็มอัตรา นั่นคือมีช่องทางสำหรับรับข้อมูลเข้า และช่องทางส่งข้อมูลออกแยกกัน ไลน์ดิสซิพรินจะอยู่ในเคอร์เนลระหว่างโปรแกรมขับอุปกรณ์แท้และโปรเซสของผู้ใช้ ดังรูปที่ 2.1



รูปที่ 2.1 แสดงไลน์ดิสชิพรินในขณะที่ใช้โปรแกรมเชลล์โดยวิธีปกติ

มีงานหลายอย่างที่อาจถูกทำได้ด้วยส่วนของไลน์ดิสชิพริน เป็นต้นว่า

- แสดงตัวอักขระที่ถูกป้อนเข้าไป
- ตรวจสอบตัวอักขระที่ป้อนเข้าเพื่อเก็บไว้เป็นบรรทัด ซึ่งโปรเซสต่างๆ จะมาอ่านข้อมูลจากบรรทัดที่ตรวจสอบแล้ว
- แก้ไขข้อมูลในบรรทัดที่กำลังป้อนเข้า
- สร้างสัญญาณ (signal) เมื่อมีการกดปุ่มสัญญาณจากเทอร์มินัล เช่น สัญญาณ SIGINT สัญญาณ SIGQUIT เป็นต้น

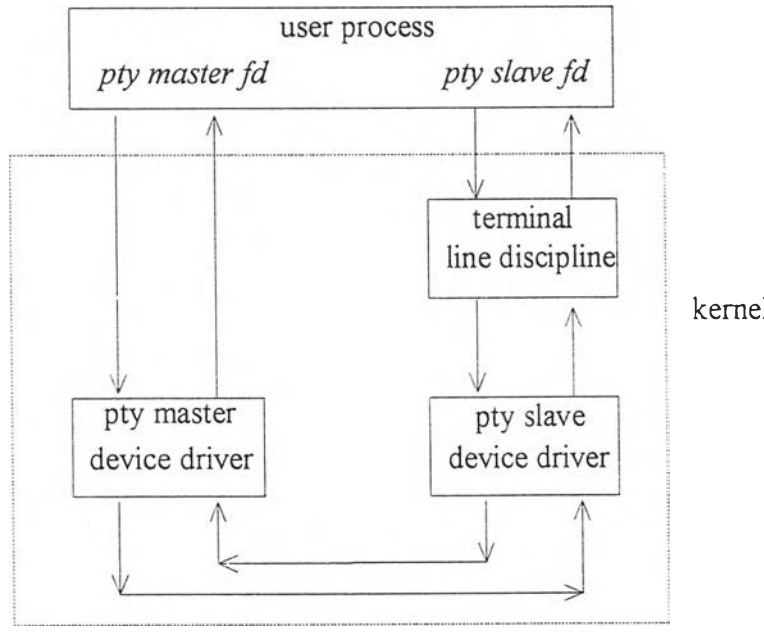
- ควบคุมการไหลของข้อมูล เช่น เมื่อผู้ใช้กดคอนโทรล-เอส (control-S) ข้อมูลที่ถูกส่งออกทางเทอร์มินัลจะถูกหยุดการแสดงผลที่จะออกมาทางจอภาพ และเมื่อผู้ใช้ต้องการให้การแสดงผลดำเนินต่อให้กดคอนโทรล-คิว (control-Q) เป็นต้น

- ยอมให้ผู้ใช้ป้อนตัวอักขระของการสิ้นสุดแฟ้มข้อมูล

- เปลี่ยนแปลงตัวอักขระบางตัว เช่น ทุกครั้งที่โปรเซสส่งตัวอักขระนิวไลน์ (newline) ไลน์ดิสซิพรินสามารถเปลี่ยนให้เป็นการย้ายการแสดงผลไปที่ต้นบรรทัดของบรรทัดถัดไป เป็นต้น

เนื่องจากมีโปรแกรมบางโปรแกรมที่ต้องใช้เทอร์มินัลในการรับ-ส่งข้อมูล แต่มีบางครั้งที่จำเป็นต้องเปลี่ยนช่องทางการรับ-ส่งข้อมูลเนื่องจากไม่สามารถใช้เทอร์มินัลได้ ดังนั้นจึงมีผู้คิดทำเทอร์มินัลเทียมขึ้นเพื่อแก้ไขปัญหาในลักษณะดังกล่าว

เทอร์มินัลเทียมเป็นอุปกรณ์คู่ประกอบด้วยมาสเตอร์(master) และสเลฟ (slave) โปรเซสจะต้องเปิดเทอร์มินัลเทียมเป็นคู่ดังกล่าวก่อนที่จะใช้งาน ส่วนที่เป็นสเลฟจะเป็นส่วนที่เชื่อมต่อกับโปรเซสของผู้ใช้ซึ่งจะเป็นเสมือนกับเทอร์มินัลปกติ ดังรูปที่ 2.2



รูปที่ 2.2 แสดงเทอร์มินัลเทียม

ไคล์เอ็นต์-เซอร์ฟเวอร์ (Client-Server)

รูปแบบที่นิยมใช้ในการพัฒนาโปรแกรมเกี่ยวกับระบบเครือข่ายแบบหนึ่ง คือ ไคลเอ็นต์-เซอร์ฟเวอร์ โดยเซอร์ฟเวอร์จะเป็นโปรเซสที่คอยการติดต่อจากไคลเอ็นต์เพื่อให้บริการบางอย่างแก่ไคลเอ็นต์ ลักษณะการทำงานของรูปแบบไคล์เอ็นต์-เซอร์ฟเวอร์โดยทั่วไปเป็นดังนี้

1. เมื่อเซอร์ฟเวอร์กำหนดค่าเริ่มต้นต่างๆ เพื่อเตรียมทำงานเสร็จแล้ว จะหยุดการทำงานเพื่อรอการติดต่อจากไคล์เอ็นต์

2. ไคล์เอ็นต์โปรเซสจะถูกปฏิบัติงานในระบบเดียวกัน หรือบนระบบอื่นที่สามารถติดต่อกับเซอร์ฟเวอร์ได้ด้วยระบบเครือข่าย ไคล์เอ็นต์โปรเซสจะส่งคำขอบริการผ่านระบบเครือข่าย เพื่อขอบริการต่างๆ จากเซอร์ฟเวอร์

3. เมื่อเซอร์ฟเวอร์ได้รับคำร้องขอจากไคล์เอ็นต์ และให้บริการแก่ไคล์เอ็นต์แล้ว เซอร์ฟเวอร์จะกลับเข้าสู่ภาวะรอการติดต่อจากไคล์เอ็นต์อีก

เซอร์ฟเวอร์โปรเซสอาจแบ่งได้เป็น 2 ประเภท ดังนี้

1. เมื่อเซอร์ฟเวอร์ให้บริการแก่ไคล์เอ็นต์ตามคำขอด้วยตัวเอง ซึ่งมักใช้เวลาสั้นๆ เรียกว่า อิเทอเรทีฟ เซอร์ฟเวอร์ (iterative server)

2. เมื่อเซอร์ฟเวอร์ไม่ทราบแน่ว่าจะต้องใช้เวลามากน้อยเท่าใดในการให้บริการแก่ไคล์เอ็นต์ ซึ่งปกติแล้วเซอร์ฟเวอร์จะใช้วิธีให้บริการแบบพร้อมกัน(concurrent) จึงเรียกว่า คอนเคอเร็นท์ เซอร์ฟเวอร์ (concurrent server) ทำได้โดยการที่เซอร์ฟเวอร์จะสร้างโปรเซสใหม่อีก 1 โปรเซส ที่จะให้เป็นตัวจัดการให้บริการแก่ไคลเอ็นต์แทน เพื่อที่เซอร์ฟเวอร์จะได้กลับไปรอรับการติดต่ออื่นๆ อีก โดยทั่วไปแล้วเซอร์ฟเวอร์ประเภทนี้มักต้องทำงานบนระบบปฏิบัติการที่ยอมให้โปรเซสหลาย ๆ โปรเซสทำงานไปพร้อมกันได้

เซอร์ฟเวอร์มีขั้นตอนการทำงานโดยคร่าว ๆ ดังนี้

1. เปิดช่องการสื่อสาร

2. คอยรับการติดจต่อจากไคล์เอ็นต์

3. สำหรับอิเทอเรทีพ เซอร์ฟเวอร์ มักใช้ในกรณีที่การให้บริการสามารถทำได้ด้วยหนึ่งคำสั่งจากเซอร์ฟเวอร์ สำหรับ คอนเคอเร็นท์ เซอร์ฟเวอร์ จะสร้างโปรเซสใหม่เพื่อให้บริการแก่ไคล์เอ็นต์ ซึ่งโปรเซสใหม่นี้จะไม่สนใจการติดต่อจากไคล์เอ็นต์อื่นๆ และเมื่อให้บริการเสร็จ จะเลิกการทำงานโดยปิดช่องการสื่อสารกับไคล์เอ็นต์นั้น

4. กลับไปทำข้อ 2

ระหว่างขั้นตอนดังกล่าว ระบบจะจัดคิวอย่างใดอย่างหนึ่งของการขอการติดต่อจากไคล์เอ็นต์ที่มาถึงในขณะที่เซอร์ฟเวอร์กำลังให้บริการแก่ไคล์เอ็นต์อื่นอยู่

ไคล์เอ็นต์โปรเซสมีการทำงานโดยทั่วไป ดังนี้

1. เปิดช่องการสื่อสาร และขอติดต่อไปยังเซอร์ฟเวอร์

2. ส่งข้อความการขอบริการไปยังเซอร์ฟเวอร์ และรับการให้บริการ ทำเช่นนี้เรื่อยไปเท่าที่ต้องการ

3. ปิดช่องการสื่อสารกับเซอร์ฟเวอร์

ซอกเกตไลบรารีฟังก์ชัน (Socket Library Functions)

สำหรับตัวประสานโปรแกรมประยุกต์การสื่อสาร (communication APIs, application program interfaces) ที่ใช้กันอย่างมากบนระบบยูนิกซ์ คือ Berkeley socket interface และ System V Transport Layer Interface (TLI) ซึ่งทั้งสองตัวประสานได้ถูกพัฒนาเพื่อใช้กับภาษาซี ในบทนี้จะได้กล่าวถึงการใช้ชุดคำสั่งซอกเกตโดยสังเขป

1. ซอกเกตแอดเดรส (Socket Address)

ระบบเครือข่ายแบบบีเอสดีทั่วไปมักต้องการตัวชี้ที่ชี้ไปยังโครงสร้างของซอกเกตแอดเดรส เพื่อเป็นอาร์กูเมนต์ นิยามของโครงสร้างนี้ถูกกำหนดไว้ใน <sys/socket.h> ดังนี้

```
struct sockaddr {
    u_short  sa_family; /* address family AF_xxx value */
    char  sa_data[14];  /* up to 14 bytes of protocol-
                           specific address */
};
```

ส่วนของหน่วยความจำที่จองไว้ 14 ไบท์ สำหรับแอดเดรสของโปรโตคอลที่กำหนดจะถูกแปลความตามชนิดของแอดเดรส

สำหรับอินเตอร์เนทแฟมิลี่ (Internet family) ได้กำหนดโครงสร้างไว้ใน ‹netinet/in.h› ดังนี้

```
struct in_addr {
     u_long  s_addr;  /* 32-bit netid/hostid */
                      /* network byte ordered */
};

struct sockaddr_in {
     short  sin_family;          /* AF_INET */
     u_short  sin_port;          /* 16-bit port number */
                                 /* network byte ordered */
     struct in_addr  sin_addr;   /* 32-bit netid/hostid */
                                 /* network byte ordered */
     char  sin_zero[8];          /* unused */
};
```

การกำหนดชื่อเพื่อใช้แทนประเภทข้อมูลแบบจำนวนไร้เครื่องหมายในระบบบีเอสดี และระบบชิสเต็มไฟว์มีความแตกต่างกัน ซึ่งจะต้องดึง ‹sys/types.h› เข้ามารวมในโปรแกรมโดยกำหนดไว้ดังนี้

C data type	4.3BSD	System V
unsigned char	u_char	unchar
unsigned short	u_short	ushort
unsigned int	u_int	uint
unsigned long	u_long	ulong

สำหรับยูนิกซ์โดเมน (Unix Domain) ได้กำหนดโครงสร้างของแอดเดรสไว้ใน ‹sys/un.h› ดังนี้

```
struct sockaddr_un {
     short  sun_family;    /* AF_UNIX  */
     char  sun_path[108];  /* pathname */
};
```

2. การใช้ Socket System Calls เบื้องต้น

2.1 Socket System Calls

การจัดการเกี่ยวกับอินพุทและเอาพุทในระบบเครือข่าย สิ่งแรกที่โปรเซสต้องทำ คือ การเรียกใช้ฟังก์ชัน socket โดยกำหนดประเภทของการติดต่อตามต้องการ เช่น Internet TCP Internet UDP XNS SPP เป็นต้น ดังนี้

```
#include ‹sys/types.h›
#include ‹sys/socket.h›

int socket(int family, int type, int protocol);
```

โดยที่

ค่าของ family ได้แก่

AF_UNIX	Unix internal protocols
AF_INET	Internet protocols
AF_NS	Xerox NS protocols
AF_IMPLINK	IMP link layer

ค่าของ type ได้แก่

SOCK_STREAM	stream socket
SOCK_DRAM	datagram socket
SOCK_RAW	raw socket
SOCK_SWQUENCE	sequenceed packet socket
SOCK_RDM	reliably delivered message socket (not implemented yet)

สำหรับโปรแกรมประยุกต์ทั่วไปควรกำหนดให้ค่าของ protocol เป็น 0

ฟังก์ชัน socket จะส่งค่าจำนวนเต็มที่คล้ายกับตัวบอกแฟ้ม (file descriptor) กลับไปยังผู้เรียก ซึ่งจะเรียกค่านี้ว่า ตัวบอกซอกเกต (socket descriptor) หรือ sockfd การที่จะได้ตัวบอกซอกเกต ควรกำหนดแอดเดรสแฟมิลี่ ประเภท และโปรโตคอลให้เหมาะสมดังตารางต่อไปนี้

	AF_UNIX	AF_INET	AF_NS
SOCK_STREAM	Yes	TCP	SPP
SOCK_DGRAM	Yes	UDP	IDP
SOCK_RAW		IP	Yes
SOCK_SEQPACKET			SPP

ตารางที่ 2.1 แสดงความสัมพันธ์ของ แฟมิลี่ และประเภท ของซอกเกต

2.2 Bind System Call

ฟังก์ชันระบบ bind ใช้กำหนดชื่อให้กับซอกเกตที่ยังไม่มีชื่อ ดังนี้

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *myaddr,
         int addrlen);
```

อาร์กูเมนต์ตัวที่ 2 คือ ตัวชี้ที่ชี้ไปยังที่อยู่ของโปรโตคอลที่กำหนดไว้และอาร์กูเมนต์ที่ 3 คือขนาดของโครงสร้างที่อยู่ การใช้ฟังก์ชันระบบ bind มีอยู่ 3 ลักษณะคือ

2.2.1 เซอร์ฟเวอร์แจ้งแอดเดรสให้กับระบบทราบเพื่อบอกระบบว่าสำหรับข่าวสารใดๆ ที่ได้รับทางแอดเดรสนี้ให้ส่งมาให้เซอร์ฟเวอร์ด้วย ทั้งเซอร์ฟเวอร์แบบคอนเนคชันโอเรนเทด (connection-oriented) และแบบคอนเนคชันเลส (connection-less) ต้องทำข้อนี้ก่อนที่จะรับการติดต่อจากไคลเอนต์

2.2.2 ไคลเอนต์ใช้กำหนดแอดเดรสสำหรับไคลเอนต์เอง

2.2.3 ไคลเอนต์แบบคอนเนคชันเลสต้องการแน่ใจว่าระบบได้กำหนดแอดเดรสที่ไม่ซ้ำกับโปรเซสอื่นให้ เพื่อที่เซอร์ฟเวอร์จะสามารถตอบรับได้ถูกต้อง

2.3 Connect System Call

ฟังก์ชันระบบ connect ใช้เมื่อไคลเอนด์ต้องการติดต่อกับเซอร์ฟเวอร์

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *servaddr,
        int addrlen);
```

sockfd คือ ตัวบอกซอกเกต ที่ได้มาจากการเรียกใช้ฟังก์ชันระบบ socketส่วนอาร์กูเมนต์ที่ 2 และ 3 เป็นตัวชี้ที่ชี้ไปยังโครงสร้างแอดเดรสของซอกเกต และขนาดของแอดเดรส ตามลำดับ สำหรับโปรโตคอลที่ใช้กับคอนเนคชันโอเรนเทด เช่น TCP SPP เป็นต้น โดยส่วนใหญ่ฟังก์ชันระบบ connect จะสร้างเส้นการติดต่อระหว่างระบบท้องถิ่น (local system) และระบบอื่น (foreign system) โดยที่ไคลเอนด์ไม่ต้องผูก (bind) local address ก่อนเรียกใช้ฟังก์ชันระบบ connect

2.4 Listen System Call

ฟังก์ชันระบบ listen ถูกใช้โดยเซอร์ฟเวอร์แบบคอนเนคชันโอเรนเทด เพื่อแสดงว่ากำลังรอรับการติดต่อ

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

โดยปกติฟังก์ชันระบบนี้จะถูกเรียกใช้ทันทีหลังจากที่เรียกใช้ฟังก์ชันระบบ socket และ ฟังก์ชันระบบ bind ก่อนที่จะเรียกใช้ฟังก์ชันระบบ accept อาร์กูเมนต์

backlog เป็นตัวกำหนดจำนวนของการขอติดต่อที่สามารถอยู่ในคิว (queue) โดยระบบจะช่วยดูแลให้ในระหว่างที่เซอร์ฟเวอร์กำลังเรียกใช้ฟังก์ชันระบบ accept โดยทั่วไปอาร์กูเมนต์นี้จะถูกกำหนดให้มีค่าเป็น 5 ซึ่งเป็นค่าที่มากที่สุดที่สามารถใช้ได้ในปัจจุบัน

2.5 Accept System Call

หลังจากที่เซอร์ฟเวอร์แบบคอนเนคชันโอเรนเทดได้ทำคำสั่งฟังก์ชันระบบ listen ดังที่ได้กล่าวข้างต้น เส้นทางการติดต่อที่ไคล์เอนต์ได้ขอติดต่อมาจะยังใช้การไม่ได้จนกว่าเซอร์ฟเวอร์จะทำคำสั่งฟังก์ชันระบบ accept เสร็จ

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *peer,
        int *addrlen);
```

ฟังก์ชันระบบ accept จะรับการติดต่อเพื่อสร้างเส้นทางการติดต่อกับไคล์เอนต์โดยการสร้างซอกเกตที่มีคุณสมบัติทุกประการเหมือนกับ sockfd ขึ้นมา และถ้าไม่มีไคล์เอนต์ใดมาขอติดต่อด้วย การเรียกใช้ฟังก์ชันระบบ accept จะต้องหยุดรอจนกว่าจะมีการติดต่อมาอาร์กูเมนต์ peer จะถูกใช้เพื่อส่งที่อยู่ของไคล์เอนต์ที่ติดต่อด้วย ส่วนอาร์กูเมนต์ addrlen จะถูกใช้เก็บขนาดของที่อยู่ที่เก็บอยู่ในอาร์กูเมนต์ peer ซึ่งทั้ง 2 อาร์กูเมนต์จะถูกส่งกลับไปให้ผู้เรียกใช้ฟังก์ชันนี้

สมมติว่าเซอร์ฟเวอร์เป็นแบบคอนเคอร์เร็นท์เซอร์ฟเวอร์ การใช้ฟังก์ชันระบบนี้ โดยทั่วไปมักอยู่ในลักษณะต่อไปนี้

```
int sockfd, newsockfd;

if ((sockfd = socket(...)) < 0)
    err_sys("socket error");
if (bind(sockfd, .. ) < 0)
    err_sys("bind error");
if (listen(sockfd, 5) < 0)
    err_sys("listen error");

for (;;) {
    newsockfd = accept(sockfd, ... );    /* block */
    if (newsockfd < 0)
        err_sys("accept error");

    if (fork() == 0) {
        close(sockfd);    /* child */
        doit(newsockfd);  /*  process  the request */
        exit(0);
    }
    close(newsockfd);     /* parent */
}
```

เมื่อมีการติดต่อมายังเซอร์ฟเวอร์และเซอร์ฟเวอร์ยอมรับการติดต่อแล้ว เซอร์ฟเวอร์จะทำการออกลูก(fork) เพื่อให้โปรเซสลูกคอยให้บริการแก่การติดต่อนั้น ส่วนเซอร์ฟเวอร์จะรอรับการขอการติดต่ออื่นๆ ต่อไป

สำหรับกรณีเซอร์ฟเวอร์แบบอิเทอเรทีฟเซอร์ฟเวอร์ โดยทั่วไปมักอยู่ในลักษณะต่อไปนี้

```
int sockfd, newsockfd;

if ((sockfd = socket( ... )) < 0)
    err_sys("socket error");
if (bind(sockfd, ... ) < 0)
    err_sys("bind error");
if (listen(sockfd, 5) < 0)
    err_sys("listen error");

for (;;) {
    newsockfd = accept(sockfd, ... );    /* block */
    if (newsockfd < 0)
        err_sys("accept error");

    doit(newsockfd);        /* process the request */
    close(newsockfd);
}
```

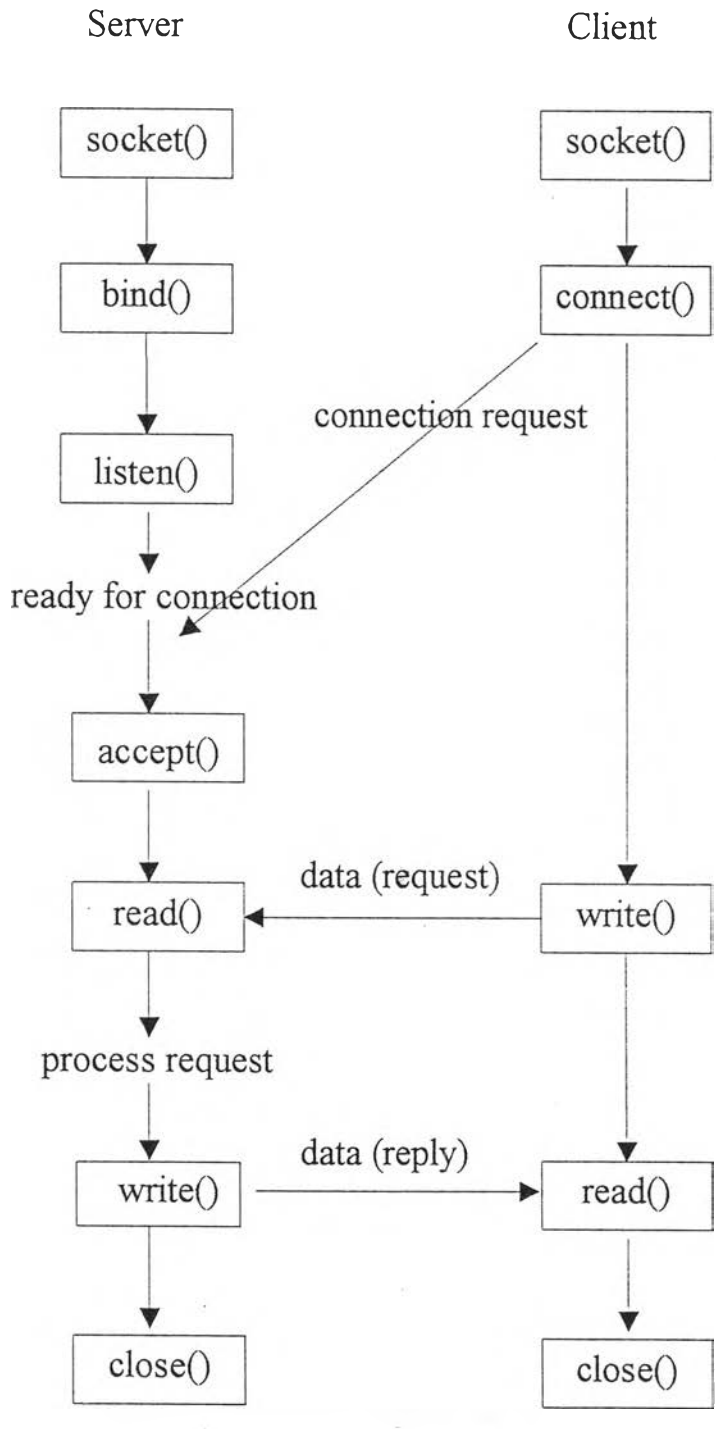
2.6 Close System Call

ฟังก์ชัน close ของระบบยูนิกซ์สามารถใช้เพื่อปิดซอกเกตได้

```
int close(int fd);
```

โดยปกติระบบจะกลับจากการทำฟังก์ชัน close ทันที ซึ่งถ้าซอกเกตที่กำลังถูกปิดมีความเกี่ยวข้องกับโปรโตคอลที่มีต้องทำการส่งอย่างเชื่อถือได้ (reliable delivery) ตัวอย่างเช่น TCP หรือ SPP เป็นต้น หลังจากฟังก์ชันระบบ close ทำเสร็จ ระบบจะต้องจัดการกับข้อมูลที่ยังคงค้างอยู่ในเคอร์เนลให้เรียบร้อย

สำหรับวิทยานิพนธ์ได้เลือกใช้ซอกเกตแอดเดรสแบบยูนิกซ์โดเมน เพราะโปรเซสต่างๆ ต้องทำงานบนเครื่องเดียวกัน และใช้เซอร์เวอร์แบบอิเทอเรทีฟ ดังรูปที่ 2.3 ซึ่งเป็นตัวอย่างบางส่วนของการใช้ชุดคำสั่งซอกเกตในวิทยานิพนธ์นี้



รูปที่ 2.3 แสดงการใช้ชุดคำสั่งซอกเกต

เทอมแคป เทอมอินโฟ และเคอร์ส (termcap terminfo and curses)

สำหรับเทอร์มินัลที่ใช้ในระบบยูนิกซ์มักมีคุณสมบัติหรือความสามารถพิเศษที่แตกต่างกันตามแต่ละบริษัทจะผลิตออกมา บางโปรแกรมอาจไม่ต้องการคุณสมบัติเหล่านั้น เช่น cat หรือ ls เป็นต้น แต่สำหรับโปรแกรมประเภทบรรณาธิกร เช่น vi เป็นต้น มักมีความต้องการคุณสมบัติพิเศษ เพื่อที่จะสามารถทำงานได้ดีขึ้น ดังนั้นเพื่อที่จะให้โปรแกรมต่างๆ สามารถที่จะนำคุณสมบัติเหล่านั้นไปใช้ได้อย่างมีประสิทธิภาพ บิล จอย (Bill Joy) จึงได้คิดสร้างกลไกที่จะควบคุมการทำงานของเทอร์มินัลที่มีความแตกต่างดังกล่าวขึ้นซึ่งประกอบด้วย 2 ส่วน คือ ส่วนแรกเป็นฐานข้อมูลที่บ่งบอกถึงคุณสมบัติของแต่ละเทอร์มินัล และส่วนที่สองเป็นซับรูทีน ซึ่งเป็นวิธีการที่จัดเตรียมไว้เพื่อให้โปรแกรมต่างๆ สามารถรู้ได้ว่าเทอร์มินัลต่างๆ มีคุณสมบัติอะไรบ้าง และจะสามารถใช้คุณสมบัติเหล่านั้นได้อย่างไรทั้ง 2 ส่วนนี้ถูกเรียกว่า **เทอมแคป** (termcap) ซึ่งเป็นคำเรียกอย่างสั้นของคำว่า term<u>inal cap</u>abilities.

แรกเริ่มนั้นเทอมแคปได้ถูกพัฒนาขึ้นในระบบยูนิกซ์แบบบีเอสดี ต่อมาได้มีการพัฒนาปรับปรุงเทอมแคปเพื่อใช้ในระบบยูนิกซ์แบบซิสเต็มไฟว์ เรียกว่า **เทอมอินโฟ** (terminfo) ความแตกต่างที่สำคัญระหว่างเทอมแคป และเทอมอินโฟ คือ เทอมอินโฟ เป็นฐานข้อมูลที่ต้องผ่านการแปล(compile) และถูกเก็บไว้แยกตามเทอร์มินัลแต่ละชนิดในโครงสร้างไฟล์แบบไฮราคี (directory hierarchy) ในขณะที่เทอมแคปจะเป็นแฟ้มข้อมูลเพียงแฟ้มเดียวที่มนุษย์สามารถอ่านได้ (human-readable text file) และเนื่องจาก เทอมอินโฟถูกพัฒนาโดยใช้เทอมแคปเป็นพื้นฐาน ดังนั้นเทอมอินโฟจึงมีฐานข้อมูลหรือประสิทธิภาพค่อนข้างมากกว่า แต่การศึกษาและการใช้งานจะกระทำได้ยากกว่า

เนื่องจากการพัฒนาโปรแกรมเพื่อให้สามารถใช้เทอมแคป หรือเทอมอินโฟ จากฐานข้อมูล และซับรูทีนที่มีให้โดยตรงนั้นไม่สามารถที่จะทำได้โดยง่าย เป็นต้นว่า ต้องทราบว่าโปรแกรมที่พัฒนาขึ้นนี้จะใช้กับเทอร์มินัลชนิดใดบ้าง เพื่อจะได้จัดเตรียมส่วนควบคุมการทำงานของเทอร์มินัลแต่ละชนิดไว้ ซึ่งนับว่าเป็นความยุ่งยากลำบากอย่างมากในการกระทำเช่นนั้น จึงได้มีผู้คิดทำ เคอร์ส (curses) ซึ่งเป็นวิธีการที่จะช่วยให้โปรแกรมต่างๆ สามารถใช้เทอมแคป

หรือเทอมอินโฟได้โดยไม่ต้องคำนึงถึงชนิดของเทอร์มินัล และยังสร้างวิธีการที่จะควบคุมการทำงานของเทอร์มินัลให้มีประสิทธิภาพสูง คำว่า เคอร์ส นั้นมีความหมายถึง การจัดการเกี่ยวกับตัวชี้ตำแหน่ง (cursor manipulation) ผู้สร้างเคอร์สคนแรกคือ เคน อาโนล์ด (Ken Arnold) ซึ่งมักจะพบเห็นในระบบยูนิกซ์ทั่วไป ต่อมา มาร์ค ฮอร์ตัน (Mark Horton) ที่เอทีแอนด์ที (AT&T) ได้ปรับปรุงขึ้นใหม่ซึ่งใช้ได้เฉพาะระบบยูนิกซ์แบบซิสเต็มไฟว์เท่านั้น เราเรียกเคอร์สรุ่นแรกว่า เคอร์สแบบเก่า (old curses) และเรียกเคอร์สรุ่นใหม่ว่า เคอร์สแบบใหม่ (new curses) ซึ่งสามารถตรวจสอบอย่างง่าย ๆ ว่าในระบบยูนิกซ์ที่ใช้อยู่เป็นเคอร์สแบบใดโดยดูว่ามีการกำหนดค่าคงที่ A_UNDERLINE ไว้ในเฮดเดอร์ไฟล์ของเคอร์สหรือไม่ดังตัวอย่างต่อไปนี้

```
$ grep A_UNDERLINE /usr/include/curses.h
#define A_UNDERLINE     0000400
$
```

ถ้าพบดังตัวอย่าง แสดงว่าเป็นเคอร์สแบบใหม่

วิทยานิพนธ์นี้ได้เลือกใช้เทอมแคปด้วยเหตุผลที่สำคัญคือ ความสามารถในการนำโปรแกรมไปใช้บนเครื่องอื่นได้โดยง่าย (portable) ประกอบกับมีบางส่วนของวิทยานิพนธ์ที่ได้เลือกใช้คุณลักษณะของยูนิกซ์แบบบีเอสดี ซึ่งทำให้การพัฒนาปรับปรุงวิทยานิพนธ์ได้สะดวก และจะได้อธิบายถึงการใช้เทอมแคปดังต่อไปนี้

1. ฐานข้อมูลเทอมแคป (Termcap Database)

ฐานข้อมูลเทอมแคปจะถูกเก็บเป็นแฟ้มข้อความ(text file) ไว้ที่แฟ้มที่มีชื่อว่า /etc/termcap การค้นหาคุณสมบัติของเทอร์มินัลแต่ละชนิดจะกระทำโดยการค้นหาแบบตามลำดับก่อนหลัง(sequential)เพราะว่าข้อมูลในเทอมแคปมีเป็นจำนวนมากกว่า 100,000 ไบต์ ดังนั้นจึงควรกำหนดคุณสมบัติของเทอร์มินัลชนิดที่ต้องใช้อยู่เสมอๆ ไว้ในส่วนต้นของแฟ้ม

1.1 การอ่านคุณสมบัติของเทอร์มินัลที่ถูกกำหนดไว้ในเทอมแคป

ตัวอย่างของคุณสมบัติของเทอร์มินัลที่ถูกกำหนดไว้ใน termcap

```
# incomplete termcap entry for the Wyse WY-50
n9¦wy50¦WY50¦Wyse Technology WY-50:\
   :bs:am:co#80:li#24:\
   :up=^K:cl=^Z:ho=^^:nd=^L:cm=\E=%+ %+ :
```

ตัวอักขระแบคสแรช (backslash character) ข้างท้ายของแต่ละบรรทัดนั้นมีความหมายว่ายังมีคุณสมบัติอื่นๆ อีกในบรรทัดที่ถัดมา คุณสมบัติของเทอร์มินัลแต่ละอย่างจะถูกแบ่งด้วยเครื่องหมายโคลอน (:)

ใน termcap จะมีบรรทัดอยู่ 3 แบบคือ

- บรรทัดอธิบาย (comment lines) ได้แก่ บรรทัดที่ขึ้นต้นด้วยเครื่องหมายชาร์ป (#) ดังตัวอย่าง

```
# incomplete termcap entry for the Wyse WY-50
```

- บรรทัดชื่อ (name lines) เป็นบรรทัดที่บอกชื่อหรือชนิดของเทอร์มินัล ซึ่งสามารถกำหนดชื่อเรียกหลายๆ ชื่อได้โดยคั่นด้วยเวอติคัลบาร์ (vertical bar) เช่น

n9¦wy50¦WY50¦Wyse Technology WY-50:\

- บรรทัดคุณสมบัติ (capability lines) ได้แก่ ส่วนที่เหลือจากข้างต้นเป็นบรรทัดที่ใช้บอกถึงคุณสมบัติต่างๆ ของเทอร์มินัลชนิดนั้นๆ ซึ่งต้องมีการย่อหน้าเสมอเพื่อให้แตกต่างจากบรรทัดชื่อ โดยทั่วไปมักใช้ตัวอักขระแถบ(tab character) คุณสมบัติของเทอร์มินัลจะถูกแทนด้วยตัวอักขระ 2 ตัว เพื่อใช้ในการค้นหา

1.2 คุณสมบัติของเทอร์มินัลแบ่งออกเป็น 3 ประเภทคือ

1.2.1 คุณสมบัติแบบตรรก (boolean capabilities) ประกอบด้วย ชื่อของคุณสมบัติเพียงอย่างเดียว เพื่อบ่งบอกว่าเทอร์มินัลชนิดนั้นมีคุณสมบัตินี้ เช่น

am

บ่งบอกว่าเทอร์มินัลชนิดนี้มีความสามารถทำการกั้นขวาของการแสดงตัวอักษรแบบอัตโนมัติ (automatic right margine) หมายถึง เมื่อตัวชี้ตำแหน่งเคลื่อนที่ไปถึงตำแหน่งสุดท้ายของบรรทัด ตัวชี้ตำแหน่ง(cursor) จะถูกเปลี่ยนตำแหน่งมาอยู่ที่ต้นบรรทัดถัดลงมา ซึ่งถ้าไม่มี am กำหนดไว้โปรแกรมต่าง ๆ จะสมมติว่าเทอร์มินัลชนิดนั้นไม่มีคุณสมบัติดังกล่าว

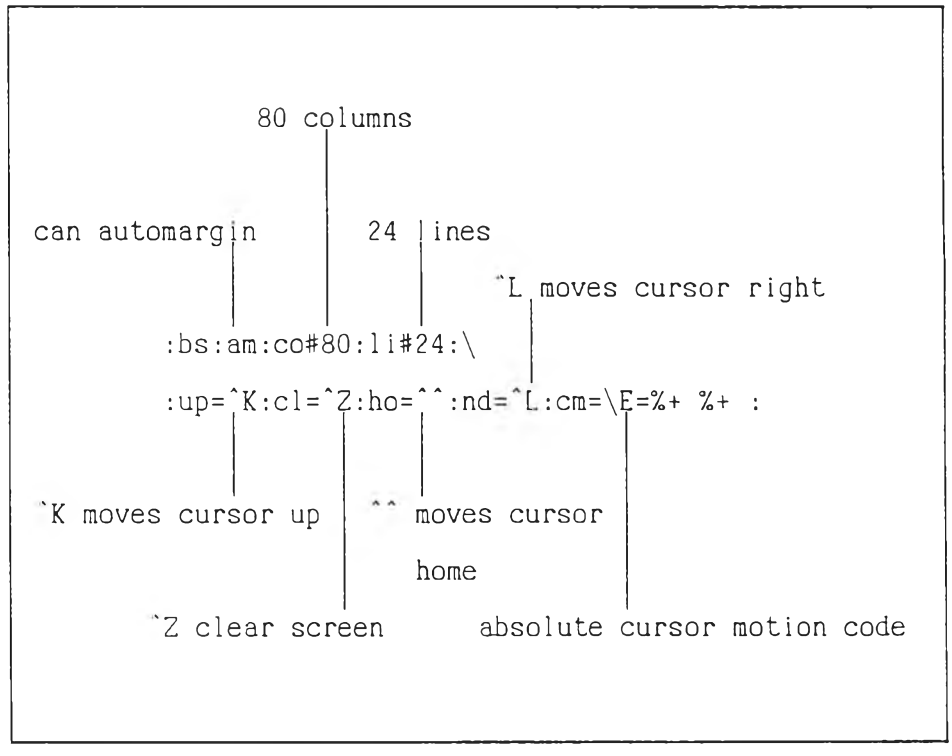
1.2.2 คุณสมบัติแบบจำนวน (numberic capabilities) ประกอบด้วย ชื่อของคุณสมบัติ เครื่องหมายชาร์ป(#) และจำนวนเต็ม เช่น

co#80

1.2.3 คุณสมบัติแบบสตริง (string capabilites) ประกอบด้วย ชื่อของคุณสมบัติ เครื่องหมายเท่ากับ(=) และลำดับคำสั่ง(command sequence) เช่น

up=^K

หมายความว่าเทอร์มินัลจะย้ายตัวชี้ตำแหน่งขึ้นไปอยู่บรรทัดก่อนหน้าบรรทัดเดิม 1 บรรทัด



รูปที่ 2.4 แสดงตัวอย่างคุณสมบัติแบบต่างๆ ของเทอมแคป

เนื่องจากข้อมูลในเทอมแคปควรจะเป็นเฉพาะตัวอักขระแอสกี้ที่สามารถแสดงหรือพิมพ์ได้เท่านั้น ดังนั้นจึงได้มีการกำหนดรหัสพิเศษเพื่อใช้แทนรหัสการควบคุมซึ่งไม่สามารถพิมพ์ได้ ดังนี้

\E	ESC
\b	backspace
\f	formfeed

\n	line feed
\r	carriage return
\t	horizontal tab
\\	\
\^	^
\nnn	charater whose code is nnn in octal
\072	:
\200	null character

คุณสมบัติแบบสตริงบางอย่างจะมีลักษณะพิเศษซึ่งมีความซับซ้อน เช่น การย้ายตัวชี้ตำแหน่ง (cm) เพราะว่าลำดับของคำสั่งไม่ใช่ค่าคงที่ขึ้นอยู่กับจำนวนแถวและสดมภ์ที่ต้องการย้ายตัวชี้ตำแหน่ง เช่น

```
cm=\E=%+ %+ :
```

หมายความว่า จำนวนแถวและสดมภ์ที่ต้องการจะถูกบวกกับค่าแอสกี้ของตัวอักขระว่าง (space character) ตัวอย่างเช่น ถ้าต้องการย้ายตัวชี้ไปที่แถวที่ 6 และสดมภ์ที่ 18 โปรแกรมจะต้องส่งลำดับของตัวอักขระ 4 ตัวดังนี้ "ESC=&2" ซึ่งสามารถคำนวณได้จาก ค่าของตัวอักขระว่าง คือ 32 ต้องการย้ายไปที่แถวที่ 6 ดังนั้น 32 + 6 = 38 ซึ่งเป็นค่าแอสกี้ของตัวอักขระ "&" และในทำนองเดียวกัน จะได้ว่า 32 + 18 = 50 ซึ่งเป็นค่าแอสกี้ของตัวอักขระ "2"

2. การใช้เทอมแคป

ดังได้กล่าวข้างต้นแล้วว่าเทอมแคปนั้นประกอบด้วยส่วนฐานข้อมูล และชับรูทีน การที่จะนำวิธีการใช้ฐานข้อมูลมาเชื่อมโยง(link) เข้ากับโปรแกรมที่พัฒนาขึ้นเองนั้น จำเป็นต้องกำหนด -l(library) เป็นอาร์กูเมนต์ของคำสั่ง cc ดังตัวอย่าง

```
$ cc main.o ps_termc.c -ltermcap
```

หรือถ้าใช้เคอร์สแบบใหม่เทอมแคปจะถูกรวมเข้ากับเคอร์ส ดังนั้นจึงควรใช้คำสั่งดังนี้

```
$ cc main.o ps_termc.c -lcurses
```

เพื่อที่จะสามารถใช้วิธีการต่างๆของเทอมแคปสิ่งแรกที่ต้องทำคือ การกำหนดตัวแปรส่วนกลางไว้ 4 ตัวแปร คือ

```
/*-----------------------------------------------------
*    External variable needed by tgoto() and tputs()
*-----------------------------------------------------*/
char *BC;      /* backspace ("bc" capability) */
char *UP;      /* cursor up ("up" capability) */
char PC;       /* pad character ("pc" capability) */
short ospeed; /* terminal speed (from ioctl()) */
```

ตัวแปร BC และ UP จะต้องถูกกำหนดให้ไปชี้ที่คุณสมบัติ bc และ up ก่อนที่จะมีการเรียกใช้ tgoto และในทำนองเดียวกันตัวแปร PC ต้องถูกกำหนดให้มีค่าเท่ากับตัวอักขระตัวแรกของ

คุณสมบัติ pc และกำหนดให้ตัวแปร ospeed มีค่าเท่ากับความเร็วของเทอร์มินัล ก่อนที่จะเรียกใช้ tputs

การใช้เทอมแคปวิธีแรกคือ การเรียกใช้ tgetent เพื่อดึงคุณสมบัติต่างๆ ของเทอร์มินัลที่ต้องการทั้งหมด

```
/*-----------------------------------------------------
*    tgetent - get Termcap entry
*    RETURN: -1 or 0 on error; 1 on success
*-----------------------------------------------------*/
int
tgetent(bp, name)
char *bp;          /* buffer to hold entry */
char *name;        /* name of terminal */
```

ตัวอย่าง

```
char tcbuf[1024], *getenv();

switch (tgetent(tcbuf, getenv("TERM"))) {
case -1:
     fatal("can't open termcap file");
case 0:
     fatal("terminal not found");
case 1:
     break;
```

```
default:
     fatal("bad return from tgetent");
}
```

และหลังจากที่ได้ใช้ tgetent จะสามารถดึงคุณสมบัติแต่ละอย่างของเทอร์มินัลนั้นได้ เช่น ใช้ tgetnum เพื่อดึงคุณสมบัติแบบจำนวน

```
/*-----------------------------------------------------
*    tgetnum - get numeric capability
*    RETURN: capability or -1 if absent
*-----------------------------------------------------*/
int
tgetnum(id)
char *id;     /* capability name */
```

ตัวอย่าง

```
int cols;

if ((cols = tgetnum("co")) == -1)
     printf("co capability is absent\n");
else
     printf("There are %d columns\n", cols);
```

ใช้ tgetflag เพื่อดึงคุณสมบัติแบบตรรกศาสตร์

```
/*-----------------------------------------------------
*    tgetflag - get Boolean capability
*    RETURN: 1 if present; 0 if absent
*-----------------------------------------------------*/
int
tgetflag(id)
char *id;     /* capability name */
```

ตัวอย่าง

```
if (tgetflag("am"))
     printf("Has automatic margin\n");
else
     printf("No automatic margin\n");
```

ใช้ tgetstr เพื่อดึงคุณสมบัติแบบสตริง

```
/*-----------------------------------------------------
*    tgetstr - get string capability
*    RETURN: pointer to capability or NULL if absent
*-----------------------------------------------------*/
char *
tgetstr(id, area)
char *id;                    /* capability name */
char **area;                 /* addredd of buffer pointer *
```

ตัวอย่าง

```
char capbuf[512], *area, *CL, *CM, *tgetstr();

area = capbuf;
CL = tgetstr("cl", &area);
CM = tgetstr("cm", &area);
```

ใช้ tgoto เพื่อสร้างลำดับของตัวอักขระสำหรับการย้ายตำแหน่งของตัวชี้ตำแหน่งตามรูปแบบของคุณสมบัติที่กำหนด

```
/*-----------------------------------------------------
*    tgoto - format cursor motion capability
*    RETURN: cursor motion sequence
*-----------------------------------------------------*/
char *
tgoto(cm, destcol, destrow)
char *cm;     /* cursor motion capability */
int destcol;  /* column */
int destrow;  /* row */
```

และใช้ tputs เพื่อส่งลำดับของตัวอักขระต่าง ๆ ไปยังเทอร์มินัล

```
/*-------------------------------------------------
*    tputs - send sequence and padding to terminal
*-------------------------------------------------*/
void
tputs(cp, affcnt, outc)
char *cp;      /* sequence to send */
int affcnt;    /* number of affected rows */
int (*outc)();/* pointer to output function */
```

ตัวอย่าง

```
static void
outc(ch)
char ch;
{
     putchar(ch);
}

char *tgoto();

tputs(CL, 1, outc);
tputs(tgoto(CM, 53, 17), 1, outc);
```

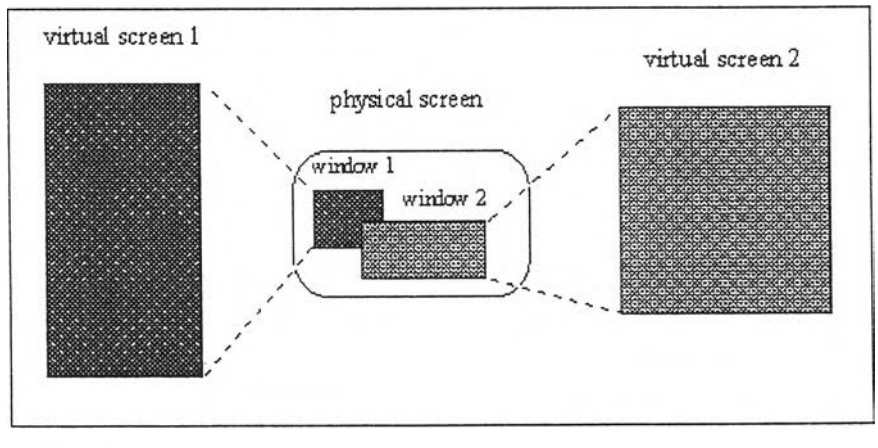
นิยามของคำศัพท์ (terminology)

เนื่องด้วยมีคำศัพท์บางคำที่ถูกใช้ในวิทยานิพนธ์นี้ได้มีผู้ให้ความหมายไว้แตกต่างกัน ซึ่งอาจเป็นสาเหตุให้มีความเข้าใจไม่ตรงกันในบางเรื่อง ดังนั้นจึงขอกำหนดความหมายของคำศัพท์ที่ใช้ในวิทยานิพนธ์ดังต่อไปนี้

จอภาพเชิงกายภาพ (physical screen) หมายถึง จอภาพที่เป็นอุปกรณ์สำหรับแสดงผลของโปรแกรม ซึ่งจะมีเพียง 1 จอภาพต่อเทอร์มินัล

หน้าต่าง (windows) หมายถึง บริเวณที่มีลักษณะเป็นรูปสี่เหลี่ยมผืนผ้าที่สามารถแสดงผลได้เช่นเดียวกับจอภาพเชิงกายภาพ แต่อาจมีขนาดเท่ากับหรือเล็กกว่า และสามารถเปลี่ยนแปลงขนาดได้

จอภาพเสมือน (virtual screen) หมายถึง บริเวณที่มีลักษณะเป็นรูปสี่เหลี่ยมผืนผ้าที่สามารถเขียนข้อมูลลงไปได้และมีขนาดใดๆ ซึ่งอาจมีขนาดใหญ่กว่า จอภาพเชิงกายภาพได้



รูปที่ 2.5 แสดงจอภาพเชิงกายภาพ หน้าต่าง และจอภาพเสมือน

ตัวอักขระ (characters) หมายถึง ตัวอักขระซึ่งโดยทั่วไปจะถูกแทนไปด้วยรหัส 8 บิท โดยจะใช้หน่วยความจำขนาด 1ไบท์ รหัสที่ใช้แทนตัวอักขระทั้งหมดมี 256 รหัส ซึ่งมีค่าตั้งแต่ 0 ถึง 255 และสำหรับตัวอักษร 128 ตัวแรก ได้ถูกกำหนดโดย American National Standard Code for Information Interchange ซึ่งเรียกสั้นๆ ตามตัวย่อว่า รหัสแอสกี (ASCII code)

ลักษณะประจำ (attributes) หมายถึง ลักษณะประจำในการแสดงผลของตัวอักขระ ได้แก่ การแสดงตัวอักขระขาวบนพื้นดำ ตัวอักขระดำบนพื้นขาว ตัวอักขระกระพริบ ตัวอักขระขีดเส้นใต้ ตัวอักขระเข้ม เป็นต้น

เซล (cells) หมายถึง วัตถุที่มีโครงสร้างที่ประกอบด้วย ตัวอักขระ และลักษณะประจำของตัวอักขระ โดยโครงสร้างดังกล่าวจะถูกกำหนดในรูปแบบของภาษาซี ดังนี้

```
typedef struct s_cell {
    char chr;                    /* character */
    char att;                    /* attribute */
} CELL;
```

เรคแทงกิล (rectangles) หมายถึง บริเวณรูปสี่เหลี่ยมผืนผ้าที่ถูกกำหนดขนาดโดยตำแหน่งของมุมบนด้านซ้ายและมุมล่างด้านขวา โดยจะถูกกำหนดในรูปแบบของภาษาซีด้วยโครงสร้างข้อมูลดังนี้

```
typedef struct s_rect {
    short r1, c1;                /* upper left corner */
    short r2, c2;                /* lower right corner */
} RECT;
```