บทที่ 4

โปรแกรมย่อยควบคุมภาษาไทย

หน้าที่ของโปรแกรมย่อยควบคุมภาษาไทย

โปรแกรมย่อยควบคุมภาษาไทยทำหน้าที่เป็นตัวเชื่อมโยงระหว่างโปรแกรมผู้ใช้กับโปรแกรมย่อยควบคุมเทอร์มินัล ในกรณีที่โปรแกรมของผู้ใช้ต้องการอ่านหรือแสดงตัวอักขระภาษาไทยบนจอภาพ ซึ่งโปรแกรมย่อยควบคุมเทอร์มินัลบนระบบปฏิบัติการยูนิกซ์ตระกูลบีเอสดี4.2 เวอร์ชั่น 2.2 นั้นทำไม่ได้

สำหรับการพัฒนาโปรแกรมย่อยควบคุมภาษาไทยบนเครื่องไมโครคอมพิวเตอร์โดยทั่วไปจะทำการแก้ไขในโปรแกรมที่ควบคุมการทำงานของระบบอินพุทเอาท์พุท (BIOS) แต่บนระบบปฏิบัติการยูนิกซ์โปรแกรมต่างๆที่เป็นส่วนประกอบของระบบปฏิบัติการจะไม่ได้รับการเปิดเผยจากบริษัทผู้ผลิต ดังนั้นแนวทางการพัฒนาจึงกระทำในระดับโปรแกรมประยุกต์ นั่นคือต้องทำการเชื่อมโยง (link) ส่วนของโปรแกรมย่อยควบคุมภาษาไทยเข้ากับโปรแกรมของผู้ใช้ ก่อนที่จะเรียกทำงาน

รูทีนของโปรแกรมย่อยควบคุมภาษาไทย

เช่นเดียวกันกับโปรแกรมย่อยควบคุมอุปกรณ์อื่นๆ โปรแกรมของผู้ใช้สามารถเรียกใช้โปรแกรมย่อยควบคุมภาษาไทยผ่านรูทีนต่างๆเหล่านี้

thopen() ทำหน้าที่ในการจัดเตรียมเทอร์มินัลให้อยู่ในสภาวะที่พร้อมจะทำงานกับตัวอักขระภาษาไทย พร้อมทั้งกำหนดค่าโดยปริยาย (default) ต่างๆ รูทีนจะนี้ต้องถูกเรียกก่อนที่จะเรียกรูทีนอื่นๆของโปรแกรมย่อยควบคุมภาษาไทย

thclose() ทำหน้าที่คืนสภาวะเดิมของเทอร์มินัลก่อนที่จะเลิกใช้โปรแกรมย่อยควบคุมภาษาไทย

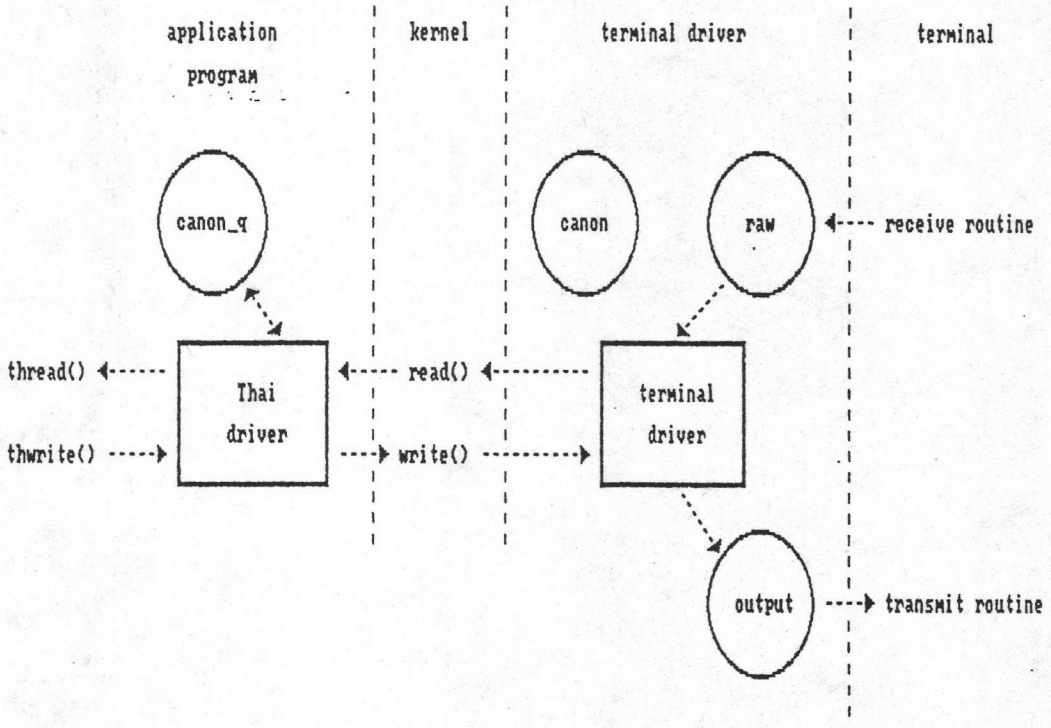
thread() ทำหน้าที่ในการอ่านข้อมูลจากแฟ้มข้อมูลหรือเทอร์มินัลแล้วส่งให้กับโปรแกรมของผู้ใช้

thwrite() ทำหน้าที่นำข้อมูลจากโปรแกรมของผู้ใช้แสดงผลออกทางจอ

ภาพหรือบันทึกลงบนแฟ้มข้อมูล

thioctl() ทำหน้าที่ในการกำหนดลักษณะข้อมูลที่เข้าและการแสดงผลบนจอภาพของตัวอักขระภาษาไทย รูทีนนี้จะมีผลต่อเมื่อมีการเรียกก่อนรูทีน thread() และ thwrite()

ลักษณะการส่งผ่านข้อมูลของโปรแกรมย่อยควบคุมภาษาไทย



รูปที่ 4.1 การไหลของข้อมูลในโปรแกรมย่อยควบคุมภาษาไทย

จากที่ได้กล่าวมาแล้วว่าโปรแกรมย่อยควบคุมภาษาไทยจะกระทำในระดับโปรแกรมประยุกต์ ดังนั้นเมื่อโปรแกรมของผู้ใช้เรียกรูทีน thread() หรือ thwrite() โปรแกรมย่อยควบคุมภาษาไทย ก็จะเรียกรูทีน read() และ write() ของโปรแกรมย่อยควบคุมอุปกรณ์ของระบบปฏิบัติการอีกที ทั้งนี้เทอร์มินัลถูกกำหนดให้อยู่ในสภาวะรอว์โหมดอยู่ก่อนแล้วจากรูทีน thopen()

เมื่อโปรแกรมของผู้ใช้เรียก thread() ในกรณีที่ต้องการอ่านแฟ้มข้อมูล โปรแกรมย่อยควบคุมภาษาไทยจะเรียกรูทีน read() แล้วส่งข้อมูลที่อ่านได้ให้กับ

โปรแกรมของผู้ใช้ แต่ถ้าเป็นการอ่านจากแป้นพิมพ์โปรแกรมย่อยควบคุมภาษาไทยจะหลับจนกว่ามีการป้อนข้อมูลเข้า ซึ่งโปรแกรมย่อยควบคุมเทอร์มินัลจะส่งสัญญาณมาบอกทำให้โปรแกรมย่อยควบคุมภาษาไทยถูกปลุกขึ้นมาทำงาน โดยอ่านข้อมูลทั้งหมดที่มีอยู่ในรอว์บัฟเฟอร์เก็บลงในคานอนคิว (ในกรณีที่ผู้ใช้กำหนดสภาวะของเทอร์มินัลให้เป็นคาโนนิคัลโหมด) พร้อมทั้งแสดงผลและจัดระดับ(สำหรับตัวอักขระภาษาไทย) ตัวอักษรเหล่านั้นบนจอภาพผ่านรูทีน write() หลังจากจัดการกับข้อมูลที่อ่านเข้ามาจนหมดโปรแกรมย่อยควบคุมภาษาไทยก็จะหลับอีก รอจนกว่ามีการส่งสัญญาณมาอีกจึงจะถูกปลุกให้ขึ้นมาทำงาน เป็นอย่างนี้เรื่อยไปจนกระทั่งข้อมูลที่อ่านเข้ามาคือ ตัวปิดแคร่จึงส่งข้อมูลที่มีอยู่ทั้งหมดในคานอนคิวให้กับโปรแกรมของผู้ใช้

สำหรับรูทีน thwrite() ในกรณีที่ต้องการบันทึกข้อมูลลงบนแฟ้มข้อมูลก็เรียกรูทีน write() แต่ถ้าหากเป็นการแสดงผลบนจอภาพ ก็จะจัดระดับตัวอักษรผ่านรูทีน write()

<u>อัลกอริทึมของโปรแกรมย่อยควบคุมภาษาไทย</u>

เนื่องจากโปรแกรมย่อยควบคุมภาษาไทยประกอบด้วยรูทีนที่โปรแกรมของผู้ใช้สามารถเรียกได้ คือ

thopen() thclose() thread() thwrite() thioctl()

จึงขอกล่าวถึงอัลกอริทึมของรูทีนเหล่านี้

อัลกอริทึมของ thopen()

```
thopen()
{
    if (check standard input and standard output are the
        same terminal) {
            set terminal into raw mode
            enable sigio for standard output
        }
    else {
        if  (standard input is terminal) {
            set terminal into raw mode
            enable sigio for standard input
        }
        /* in case redirect output to other terminal */
        if  (standard  output  is  terminal) {
            set terminal into raw mode
            enable sigio for standard output
            download thai font
        }
    }
    get_termcap() /* get escape sequence from /etc/termcap for
                     cursor movement */
    get_th_env()  /* get thai environment variable and set up
                     echo  and  display  routine  and  load
                     all table into progarm*/
}
```

thopen() เป็นรูทีนแรกของโปรแกรมย่อยควบคุมภาษาไทยที่ต้องเรียกใช้ก่อนรูทีนอื่นๆ เพื่อเตรียมเทอร์มินัลและโปรแกรมให้พร้อมก่อนที่จะรับหรือส่งข้อมูล สำหรับยูนิกซ์ไลปรารีอินพุทเอาท์พุทมาตรฐานจะกำหนดแฟ้มข้อมูลให้สำหรับแต่ละโปรเซสด้วยกัน 3 แฟ้มคือ standard input, standard output และ standard error ซึ่งในขณะเรียกใช้โปรแกรมหนึ่งๆให้ทำงานนั้น เราสามารถเปลี่ยนทิศทางของแฟ้มข้อมูลมาตรฐานเหล่านี้ได้ เช่น cat /etc/passwd > /dev/tty01 เป็น

การแสดงแฟ้มข้อมูล /etc/passwd ให้ออกที่เทอร์มินัลที่มีชื่อว่า /dev/tty03 แทนการให้แสดงที่เทอร์มินัลที่กำลังทำงานอยู่ ดังนั้นใน thopen() จึงต้องตรวจสอบว่า standard input และ standard output เป็นเทอร์มินัลเดียวกันหรือไม่ ถ้าหากใช่ก็ทำการจัดเตรียมเทอร์มินัลให้อยู่ในสภาวะรอว์โหมด และกำหนดให้มีการส่งสัญญาณ SIGIO ไปยังโปรเซสเมื่อเกิดอินพุทเอาท์พุทขึ้น เช่น เมื่อมีข้อมูลเข้ามาให้อ่านก็จะส่งสัญญาณ SIGIO มาบอก แต่ถ้าหากไม่ใช่เทอร์มินัลเดียวกันก็จะตรวจสอบว่า standard input เป็นเทอร์มินัลหรือไม่ ซึ่งถ้าหากไม่ใช่ก็จะเป็นแฟ้มข้อมูลธรรมดาจึงไม่จำเป็นต้องจัดเตรียมเทอร์มินัล สำหรับ standard output นั้นในกรณีที่เป็นเทอร์มินัลจะทำการดาวน์โหลดฟอนต์ภาษาไทยให้ด้วย (การที่ไม่ดาวน์โหลดฟอนต์ภาษาไทยลง standard input เพราะผู้ใช้ต้องเรียกใช้โปรแกรม thdl ก่อน ซึ่งสั่งเพียงครั้งเดียวก็ใช้ฟอนต์ภาษาไทยได้ตลอด นอกจากว่าจะปิดเปิดเทอร์มินัล หรือรีเซ็ทเทอร์มินัล ฟอนต์จึงจะหายไป แต่สำหรับ standard output ในกรณีที่ส่งข้อความภาษาไทยไปเทอร์มินัลอื่น ผู้ใช้เทอร์มินัลนั้นไม่อาจรู้ก่อนหรือเตรียมตัวก่อนที่จะรับข้อความภาษาไทย จึงต้องจัดเตรียมและดาวน์โหลดฟอนต์เตรียมเอาไว้ให้)

หลังจากที่จัดเตรียมเทอร์มินัลเรียบร้อยแล้วก็เรียกรูทีน get_termcap() เพื่อทำการดึงเอาคอนโทรลโคดต่างๆที่ใช้ในการเลื่อน หรือเคลื่อนย้ายตำแหน่งของเคอร์เซอร์จาก /etc/termcap ขึ้นมา

ส่วนรูทีน get_th_env() ทำการดึงค่าของตัวแปรเอนไวรอนเมนท์ THIN THOUT THMODE THCODE สำหรับ THIN หลังจากที่ได้ค่าของ THIN แล้วนั้นจะกำหนดตัวแปร int(*func_echo) ให้เก็บตำแหน่งรูทีนที่แสดงตัวอักษรในขณะที่ข้อมูลถูกป้อนเข้ามา (ในขณะเรียก thread()) รูทีนที่ใช้แสดงตัวอักษรมีอยู่ด้วยกัน 4 รูทีนคือ

out_char_1() สำหรับจัดแบบ 1 ระดับ

out_char_2() สำหรับจัดแบบ 2 ระดับ

out_char_3() สำหรับจัดแบบ 3 ระดับ

out_char_4() สำหรับจัดแบบ 4 ระดับ

ส่วน THOUT ก็กำหนดตัวแปร int (*func_out)() เก็บตำแหน่งของรูทีนที่แสดงตัวอักษร (ในขณะที่เรียก thwrite()) ซึ่งมี 5 รูทีน แต่ 4 รูทีนแรกเป็นรูทีนเดียวกันกับของ THIN ส่วนอีกรูทีนคือ out_char_pr() สำหรับจัดแบบพิมพ์ออกเครื่องพิมพ์ ซึ่งค่าของ func_echo() กับ func_out() จะเก็บตำแหน่งของรูทีนไหนขึ้นอยู่กับค่าของ THIN กับ THOUT

(หมายเหตุ thopen() จะจัดเตรียม standard input กับ standard output ที่เป็นเทอร์มินัลเท่านั้น ดังนั้นถ้าเป็นแฟ้มข้อมูลอื่นๆ ผู้ใช้ต้องเรียกฟังก์ชั่น open() เอง)

อัลกอริทึมของ thclose()

```
thclose()
{
    if (check standard input and standard output are
        the same terminal) {
        set standard input into cook mode
        disable SIGIO
    }
    else
        if (check standard input is terminal) {
            set standard input into cook mode
            disable SIGIO
        }
        if (check standard output is terminal) {
            set standard output into cook mode
            disable SIGIO
        }
}
```

thclose() ผู้ใช้จะเรียกหลังจากที่ไม่ต้องการใช้โปรแกรมย่อยควบคุมภาษาไทย thclose() จะตรวจสอบ standard input กับ standard output ว่าเป็นเทอร์มินัลเดียวกันหรือไม่ ถ้าหากใช่ก็จะเปลี่ยนสภาวะของเทอร์มินัลเป็นคาโนนิคัลโหมดซึ่งเป็นสภาวะปกติ พร้อมทั้ง disable SIGIO ด้วย แต่ถ้าหากไม่ใช่เทอร์มินัลเดียวกัน ก็จะเปลี่ยนสภาวะของแต่ละเทอร์มินัลให้เป็นคาโนนิคัลโหมดและ disable SIGIO

อัลกอริทึมของ thread()

```
thread(fd, user_buf, num)
int fd;
unsigned char *user_buf;
int num;
{
     end_flag = 0;
     if  (check fd is not terminal) {
          read all data from fd
          return
     }
     set interrupt routine is r_data (when sigio occur)
     pause()
     while (!end_flag)
          while (there are characters in buff) {
               if  (terminal is set into raw mode) {
                    put character into raw_q
                    end_flag is set to 1
               }
               else
                    /* ^C, ^Z, ^\, ^D, bs, ^U, ^R */
                    if  (character is control code) {
                         if  (character is ^D) {
                              end_flag = 1
                         }
                         process it
                    }
                    else {
                         echo character
                         put character into canon_q
                    }
                    if  (terminal is set into raw mode)
                         break;
```

```
            }
            if (end_flag = 0) pause()
        }
        if (terminal is set into raw mode) {
            copy one character from raw_q to user_buf
            return
        }
        else
            copy all characters from canon_q to user_buf
    }
```

อัลกอริทึมของ r_data()

```
    r_data()
    {
        ignore signal SIGIO
        read and kept all character into rd_buff
        while (there are any character in rd_buff) {
            if  (character is xoff)
                flag on
            else
                if  (character is xon)
                    flag off
                else
                    if  (this  routine was called while
                         working in thwrite())
                        /* do ^C, ^Z, ^/*/
                        process  each  character
                    else
                        copy each character into buff
            }
            if  (flag on) {
                read one character put into tempc
                while  (tempc is not xon) {
                    if  (tempc is not xoff) {
```

```
                    copy tempc into buff
                }
                read one character into tempc
            }
        }
        /*for new SIGIO occured */
        set interrupted routine to r_data
        return
    }
```

thread() เป็นรูทีนที่ใช้เมื่อต้องการอ่านข้อมูลแต่การทำงานจะสัมพันธ์กับรูทีน r_data ซึ่งถือได้ว่าเป็น interrupt service routine ของโปรแกรมย่อยควบคุมภาษาไทย มีหน้าที่หลักในการอ่านข้อมูลทันทีที่เกิดการขัดจังหวะโดยสัญญาณ SIGIO แล้วเก็บลงในบัฟเฟอร์ที่ชื่อ rd_buff และในกรณีที่รูทีนนี้ถูกเรียกในขณะที่ทำใน thwrite() รูทีน รูทีนนี้ก็จะทำหน้าที่จัดการกับโปรเซสด้วย

thread() จะอ่านข้อมูลจากแฟ้มข้อมูลหรือจากเทอร์มินัลขึ้นอยู่กับค่า fd ถ้าหากเป็นเพียงแฟ้มข้อมูลธรรมดาก็จะเรียกรูทีน read() ของระบบปฏิบัติการอีกทีแล้วส่งข้อมูลพร้อมทั้งจำนวนข้อมูลที่อ่านได้ให้กับโปรแกรมของผู้ใช้ แต่ถ้าหากเป็นเทอร์มินัลการส่งข้อมูลให้กับโปรแกรมของผู้ใช้จะไม่ทำทันทีที่ได้รับข้อมูลหรือทันทีที่ผู้ใช้ป้อนข้อมูลเข้ามา แต่ข้อมูลเหล่านี้จะถูกเก็บเอาไว้จนกว่าจะหมดบรรทัด นั่นคือจนกว่าจะเจอตัวอักษรขึ้นบรรทัดใหม่ (newline) กับตัวอักษรจบบรรทัด (end of file) จึงจะส่งข้อมูลให้กับโปรแกรมของผู้ใช้ ผู้ใช้เองก็ไม่ได้ป้อนข้อมูลอยู่ตลอดเวลา การทำงานของรูทีนนี้จึงทำในลักษณะการขัดจังหวะ โดยอาศัยสัญญาณ SIGIO จากระบบปฏิบัติการที่จะส่งให้เมื่อมีการป้อนข้อมูลเข้ามา ดังนั้นในรูทีน thread() จึงกำหนดรูทีน r_data ให้จัดการกับข้อมูลเมื่อเกิดการขัดจังหวะ แล้วหยุดการทำงานรอจนกว่าเกิดการขัดจังหวะขึ้น เมื่อเกิดสัญญาณ SIGIO ขึ้น การควบคุมต่างๆ จะอยู่ที่รูทีน r_data() ในรูทีน r_data() สิ่งแรกที่ต้องทำคือกำหนดให้รูทีน SIG_IGN เป็นรูทีนขัดจังหวะแทนเมื่อเกิดสัญญาณ SIGIO ขึ้น (การกำหนดเช่นนี้เป็นการไม่สนใจสัญญาณ SIGIO ที่เกิดขึ้น) ทั้งนี้เพราะว่าในขณะที่ทำงานในรูทีน r_data() ยังไม่เสร็จก็อาจเกิดสัญญาณ SIGIO ขึ้นอีกก็ได้ทำให้เกิดการเรียกรูทีน r_data() ซ้อนในขณะที่ทำ r_data() ครั้งก่อนยังไม่เสร็จ ดังนั้นในขณะที่ทำ r_data() จะไม่มีการขัดจังหวะเกิดขึ้นจนกว่าจะทำเสร็จ ใน r_data() นั้นจะทำ

หน้าที่อ่านข้อมูล แล้วกรองข้อมูลเอา xon และ xoff ออก เพื่อเอาส่วนที่เป็นข้อมูลจริงลง buff โดยที่ทั้ง xon และ xoff เป็นข้อมูลที่เกิดจากเทอร์มินัลเอง ทั้งนี้เพราะว่าในขณะที่ส่งข้อมูลไปแสดงผลบนจอภาพ ถ้าหากว่าที่พักข้อมูลบนเทอร์มินัลเกิดเต็ม ก็จะส่ง xoff มายังโปรแกรมย่อยควบคุมเทอร์มินัลฯ ก็จะเก็บลงใน raw bufffer พร้อมทั้งสัญญาณ SIGIO การทำงานของโปรแกรมย่อยควบคุมภาษาไทยในขณะนั้นจะหยุดแล้วเปลี่ยนมาทำงานที่รูทีน r_data() แทน r_data() ก็อ่าน xoff ขึ้นมา เมื่อพบว่าเป็น xoff ในขณะที่ทำรูทีน thread() r_data() ก็ยังคงอ่านข้อมูลต่อไปเรื่อยๆ จนกว่าเทอร์มินัลพร้อมที่จะรับข้อมูลก็จะส่ง xon มาให้ จึงจะคืนการควบคุมให้กับ thread() ในตำแหน่งก่อนที่จะเกิดการขัดจังหวะขึ้น ในขณะที่ได้รับ xoff มานั้นผู้ใช้อาจมีการป้อนข้อมูลเข้ามาตลอด r_data() ก็ยังคงอ่านข้อมูลเก็บลงใน buff แต่ก็มิได้ส่งออกไปแสดงผลบนจอภาพ ซึ่งก็เป็นช่วงเวลาที่เทอร์มินัลระบายข้อมูลที่อยู่บนที่พักข้อมูลออกจนสามารถที่จะรับข้อมูลใหม่จึงจะส่ง xon มา

อัลกอริทึมของ thwrite()

```
thwrite(fd, user_buf, num)
int fd;
int user_buf;
int num;
{
     if  (check fd is not terminal) {
          write all data from fd
          return
     }
     set interrupted routine to r_data
     while (character in buf is not null or
            num argument is not reached) {
               write character out
     }
```

thwrite() เป็นรูทีนสำหรับบันทึกข้อมูลลงในแฟ้มข้อมูลใดๆ หรือแสดงผลบนจอภาพ ถ้าหาก fd ที่ส่งมาให้รูทีน thwrite() หมายถึงแฟ้มข้อมูลทั่วไป ก็จะเรียกรูทีน write() ของระบบปฏิบัติการให้บันทึกข้อมูลใน user_buf จำนวน num ลงในแฟ้มข้อมูล fd แต่สำหรับกรณีที่ fd เป็นเทอร์มินัล จะกำหนดให้ r_data เป็นรูทีนที่ควบคุมการทำงานของโปรแกรมเมื่อมีสัญญาณ SIGIO เกิดขึ้น หลังจากนั้นจึงจะแสดงข้อมูลที่มีอยู่ใน user_buf ทีละตัวจนกว่าจะหมด ในขณะที่กำลังแสดงผลอยู่นั้นอาจเกิดการขัดจังหวะขึ้น ซึ่งการขัดจังหวะอาจเกิดขึ้นจาก xon และ xoff ที่ส่งมาจากเทอร์มินัลหรือที่มาจากผู้ใช้(เมื่อต้องการหยุดดูการแสดงผลเป็นระยะๆ) ลักษณะการทำงานจะเหมือนกับขณะที่เกิดขึ้นใน thread() แต่อีกกรณีเกิดขึ้นเมื่อผู้ใช้กดตัวอักษรใดๆเข้ามา ซึ่งตัวอักษรเหล่านี้อาจเป็นตัวอักษรที่ใช้ในการกำหนดการทำงานของโปรแกรม เช่น ^C ใช้เลิกการทำงาน หรือ ^Z ใช้หยุดการทำงานชั่วคราว ใน r_data เมื่อได้รับตัวอักษรเหล่านี้จะต้องทำการตรวจสอบและจัดการกับโปรเซสทันที ซึ่งแตกต่างกับรูทีน thread() ที่ไม่จัดการทันทีที่ได้รับตัวอักษรพิเศษเหล่านี้เพราะว่าตัวอักษรที่ถูกป้อนเข้ามาจะต้องถูกประมวลผลเรียงตามลำดับตามการป้อนข้อมูล เช่น เมื่อป้อนข้อมูล a b c ^C ก็จะแสดงผล a b c ออกไปก่อน จึงค่อยยกเลิกโปรเซส ดังนั้นการตรวจสอบและการจัดการกับตัวอักขรพิเศษจึงปล่อยให้เป็นหน้าที่ของ thread() เองที่ทำ ซึ่งถ้าหากปล่อยให้เป็นหน้าที่ของ r_data() การคืนการควบคุมการทำ

งานของโปรแกรมจะทำได้ช้า ซึ่งผู้ใช้ก็จะเห็นการแสดงผลช้าไปด้วย แต่สำหรับ thwrite() จะเกี่ยวข้องกับเฉพาะการแสดงผลอย่างเดียว ดังนั้นการที่ผู้ใช้ป้อนข้อมูลเข้ามาก็เพื่อที่จะควบคุมการทำงานของโปรเซส ซึ่งต้องทำทันทีเมื่อเกิดการขัดจังหวะ

อัลกอริทึมของ thioctl()

```
thioctl(func, type)
int func;
char type;
{
     switch (func)
     case SETCODE:
          change THCODE environment variable has type value
     case SETIN:
          change THIN environment variable has type value
          set new echo routine
          load thin_db table
     case SETOUT:
          change THOUT environment variable has type value
          set new writing out routine
          load thout_db table
     case SETMODE:
          change  THMODE  environment variable has type value
     case GETCODE:
          return value of THCODE environment variable
     case GETIN:
          return value of THIN environment variable
     case GETOUT:
          return value of THOUT environment variable
     case GETMODE:
          return value of THMODE environmant variable
}
```

สำหรับ thioctl() ใช้เมื่อต้องการเปลี่ยนหรือต้องการทราบลักษณะ

การทำงานของโปรแกรมย่อยควบคุมภาษาไทย ซึ่งขึ้นอยู่กับค่าของ func ซึ่งอาจมีค่าได้ 8 ค่า 4 ค่าแรกคือ SETCODE SETIN SETOUT SETMODE ใช้เพื่อเปลี่ยนลักษณะการทำงาน ส่วน GETCODE GETIN GETOUT GETMODE ใช้เพื่อต้องการทราบลักษณะการทำงานของโปรแกรมย่อยควบคุมภาษาไทยในขณะนั้น สำหรับค่าของ func 4 ค่าแรก แต่ละ func จะมีค่าเหมือนกับค่าของตัวแปรเอนไวรอนเมนท์ เช่น ค่าของ type สำหรับ func SETCODE อาจมีค่าเป็น 7 หรือ 8 เหมือน THCODE SETIN จะเหมือน THIN SETOUT จะเหมือน THOUT และ SETMODE จะเหมือน THMODE

เมื่อค่าของ func เท่ากับ SETIN จะเปลี่ยนรูทีนในการแสดงอักษรทำได้โดยกำหนดให้ func_echo (ซึ่งเป็นเพียงตัวแปรที่เก็บตำแหน่งของรูทีนการแสดงตัวอักษร) ให้ทำการเก็บตำแหน่งของรูทีนที่จะแสดงตัวอักษรใหม่ (รูทีนต่างๆมีดังนี้ out_char_1() out_char_2() out_char_3() out_char_4() และ out_char_pr()) สำหรับ func เมื่อมีค่าเท่ากับ SETOUT ก็มีการกำหนดค่าให้กับตัวแปร func_out เช่นเดียวกันกับ func_echo

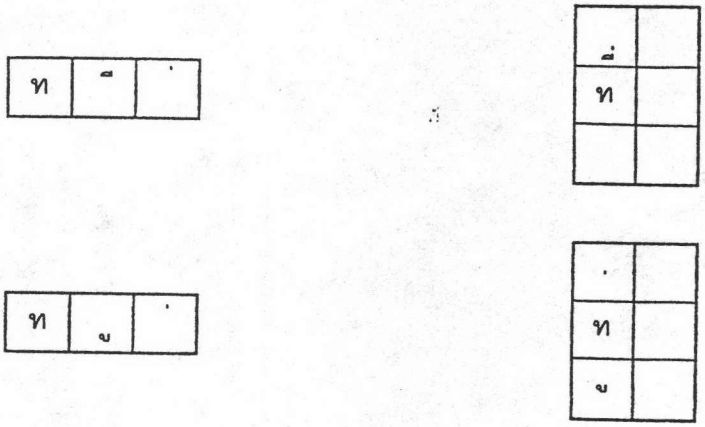
นอกจากนี้สำหรับ func ที่มีค่าเท่ากับ SETIN หรือ SETOUT จะต้องโหลดค่าจากแฟ้มข้อมูลจาก thin_db และ thout_db ลงในตารางเพราะจำนวนบรรทัดที่จะแสดงต่อภาษาไทย 1 บรรทัดเปลี่ยนไป ระดับของตัวอักขระภาษาไทยแต่ละตัวก็เปลี่ยนไปด้วย

<u>ลักษณะการแสดงภาษาไทย</u>

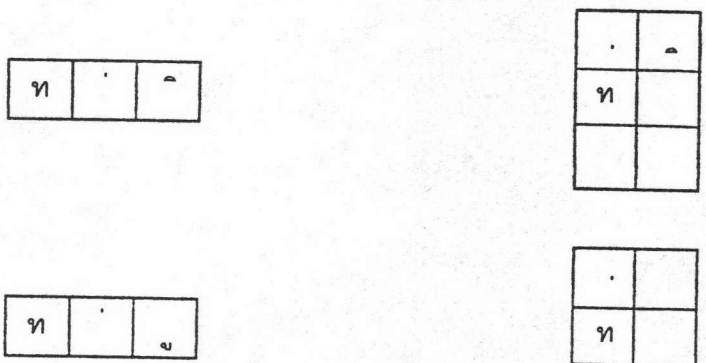
การแสดงภาษาไทยบนจอภาพวีที 220 จะต้องเตรียมเทอร์มินัลให้รับและส่งข้อมูลแบบ 8 บิต (โดยการส่งเอสเคปซีเควนซ์ไปยังเทอร์มินัล ดูได้จากคู่มือการใช้เทอร์มินัลวีที 220) สำหรับการป้อนข้อมูลเข้านั้นต้องอาศัยตัวอักขระ Ctrl-space เพื่อใช้เปลี่ยนโหมดของข้อมูลแต่ละตัวที่ถูกป้อนเข้ามามีรหัสเพียง 7 บิต ก็จะถูกนำมาแปลงเป็นรหัสของตัวอักขระไทย (สมอ.) โดยผ่านตารางที่โหลดมาจาก thin_db

สำหรับการจัดระดับอักขระไทยแบบ 2 3 4 หรือ P นั้นจะมีลักษณะเดียวกัน ดังนั้นจึงขอกล่าวถึงการแสดงการจัดแบบ 3 ระดับเท่านั้น

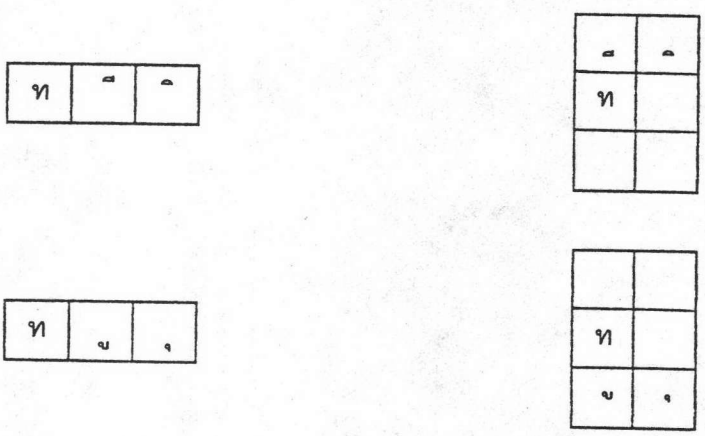
ก.



ข.



ค.



รูปที่ 4.2 การจัดระดับของตัวอักขระภาษาไทย

การแสดงตัวอักขระภาษาไทยบนจอภาพจะยึดหลักสระต้องป้อนก่อนวรรณยุกต์ (ดังเช่นภาพ ก) จึงจะมีการจัดให้อยู่ในคอลัมน์เดียวกัน แต่ถ้าวรรณยุกต์ถูกป้อนก่อนสระ จะถูกจัดให้อยู่คนละคอลัมน์ (ดังเช่นภาพ ข) นอกจากนี้การป้อนสระ

2 ตัวติดกัน สระหลังจะไม่แทนที่สระตัวหน้า แต่จะอยู่คนละคอลัมน์ (ดังเช่นภาพ ค)

อักขระภาษาไทยกับจอภาพวีที 220

การกำหนดภาษาไทยรูปแบบตัวอักษรบนจอภาพวีที 220 อาจกำหนดได้หลายรูปแบบ แต่ในขณะใดขณะหนึ่งจะต้องเลือกชุดตัวอักษรที่ต้องการลงใน in-use-table ซึ่ง in-use-table จะเป็นตารางแบบ 8 บิตที่เทอร์มินัลใช้ในการตีความตัวอักษรที่ได้รับว่าจะแสดงรูปแบบตัวอักษรอะไรออกไป ซึ่ง in-use-table แบ่งได้ออกเป็น 4 ส่วนด้วยกันคือ

รหัสตั้งแต่ 01-31 คือกลุ่ม C0 (control character set)

รหัสตั้งแต่ 32-127 คือกลุ่ม GL (left character set)

รหัสตั้งแต่ 128-159 คือกลุ่ม C1 (control character set)

รหัสตั้งแต่ 160-255 คือกลุ่ม GR (right character set)

รหัสของอักขระภาษาไทยตามที่สมอ. ได้กำหนดเริ่มตั้งแต่ 161 ขึ้นไป ดังนั้นรูปแบบของตัวอักขระภาษาไทยจึงต้องนำลงใน GR ดังนี้

8	9	10	11	12	13	14	15	COLUMN / BITS / ROW
	DCS		ฐ	ภ	ะ	เ	๐	0 0 0 0 0
	PU1	ก	ฑ	ม	ั	แ	๑	0 0 0 1 1
	PU2	ข	ฒ	ย	า	โ	๒	0 0 1 0 2
	STS	ฃ	ณ	ร	ำ	ใ	๓	0 0 1 1 3
IND	CCH	ค	ด	ฤ	ิ	ไ	๔	0 1 0 0 4
NEL	MW	ฅ	ต	ล	ี	ๅ	๕	0 1 0 1 5
SSA	SPA	ฆ	ถ	ฦ	ึ	ๆ	๖	0 1 1 0 6
ESA	EPA	ง	ท	ว	ื	็	๗	0 1 1 1 7
HTS	SOS	จ	ธ	ศ	ุ	่	๘	1 0 0 0 8
HTJ		ฉ	น	ษ	ู	้	๙	1 0 0 1 9
VTS		ช	บ	ส	ฺ	๊	ฯ	1 0 1 0 10
PLD	CSI	ซ	ป	ห		๋	๛	1 0 1 1 11
PLU	ST	ฌ	ผ	ฬ		์		1 1 0 0 12
RI	OSC	ญ	ฝ	อ		๎		1 1 0 1 13
SS2	PM	ฎ	พ	ฮ		๏		1 1 1 0 14
SS3	APC	ฏ	ฟ	ฯ	฿	๐		1 1 1 1 15
C1 CODE		GR(THAI)						

รูปที่ 4.3 รหัสของตัวอักขระภาษาไทย (สมอ.)

แต่เนื่องจากว่าการจัดระดับแบบ 2 และ 3 ระดับ จึงต้องแสดงภาพของสระบนและวรรณยุกต์อยู่ในบรรทัดเดียวกัน ดังนั้นภาพของอักขระ

่ ้ ๊ ๋ ิ ี ึ ื ็ ่ ้ ๊ ๋ ิ ี ึ ื ็

จึงต้องจัดลงในตำแหน่งดังนี้

รูปที่ 4.4 รหัสของตัวอักขระภาษาไทยที่ใช้ในการแสดงบนจอภาพ

สำหรับข้อมูลที่เป็นภาษาไทยเมื่อถูกป้อนเข้ามา (ผ่านรูทีน thread) ก็จะถูกแปลงรหัสโดยผ่านตารางที่ได้จากแฟ้มข้อมูล thin_db (ดังรูปที่ 4.5) ทำให้ได้ระดับของตัวอักขระด้วย

```
t3:
 0,0:   1,0:   2,0:   3,0:   4,0:   5,0:   6,0:   7,0
 8,0:   9,0:  10,0:  11,0:  12,0:  13,0:  14,0:  12,0
16,0:  17,0:  18,0:  19,0:  20,0:  21,0:  22,0:  23,0
24,0:  25,0:  26,0:  27,0:  28,0:  29,0:  30,0:  31,0
32,0:  44,2:  45,2:  50,2:  51,2:  52,2: 220,1: 167,2
54,2:  55,2:  53,2:  57,2: 193,2: 162,2: 227,2: 189,2
168,2: 197,2:  95,2:  47,2: 192,2: 182,2: 216,3: 214,1
164,2: 181,2: 171,2: 199,2: 204,2: 170,2: 178,2: 198,2
 49,2: 196,2:  63,2: 169,2: 174,2: 175,2: 226,2: 172,2
231,1: 179,2: 235,1: 201,2: 200,2:  46,2: 236,1: 207,2
173,2:  48,2: 177,2: 166,2: 184,2: 234,1: 206,2:  34,2
 41,2:  37,2:  40,2: 186,2:  92,2: 197,2: 217,3:  56,2
140,2: 191,2: 212,1: 225,2: 161,2: 211,2: 180,2: 224,2
233,1: 195,2: 232,1: 210,2: 202,1: 183,2: 215,1: 185,2
194,2: 230,2: 190,2: 203,2: 208,2: 213,1: 205,2: 228,2
187,2: 209,1: 188,2: 176,2: 124,2:  44,2: 126,2: 127,0
```

รูปที่ 4.5 ข้อมูลในแฟ้ม thin_db

สำหรับตัวอักษรจะเก็บลงในแอเรย์แบบหนึ่งมิติชื่อ canon_q

183	209	233	167	230	183	213	232	195	217	233
ท	ั	้	ง	ๆ	ท	ิ	่	ร	ี	้

ส่วนระดับของตัวอักษรจะเก็บลงแอเรย์หนึ่งมิติที่ชื่อ level_q

2	1	1	2	1	2	1	1	2	3	2

ซึ่ง level_q ใช้ช่วยในการพิจารณาว่าจะนิมน์รหัสของอักษรผสมหรือไม่ ถ้าหากจัดระดับแบบ 2 3 หรือ P จะสั่งเขียนรหัสต่างๆ ดังนี้

183 209 220 167 230 183 213 243 193 217 233

ท ั ้ ง ๆ ท ี ่ ร ู ้

จะเห็นว่ารหัสของตัวอักขระที่แสดงออกทางจอภาพไม่ได้เป็นรหัสของตัวอักขระที่ใช้ส่งให้กับโปรแกรมผู้ใช้

ส่วนข้อมูลภาษาไทยที่ต้องการแสดงออกทางจอภาพ (ผ่านรูทีน thwrite) จะถูกนำมาผ่านตารางที่ได้จากแฟ้มข้อมูล thout_db (ดังรูปที่ 4.6) เพื่อให้ได้ระดับของตัวอักขระเก็บลงใน level_q

```
t3:
0,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,2,2,2,2,2,
2,1,2,2,1,1,1,1,3,3,1,1,
1,1,1,0,2,2,2,2,2,0,2,1,
1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,0,
```

รูปที่ 4.6 ข้อมูลในแฟ้ม thout_db