บทที่ 3

ส่วนประกอบของโปรแกรมย่อยควบคุมภาษาไทย

โปรแกรมย่อยควบคุมภาษาไทยประกอบด้วยส่วนต่างๆของยูนิกซ์ ซึ่งจำเป็นที่ต้องกล่าวถึงมีดังต่อไปนี้

1. ตัวแปรเอนไวรอนเมนท์ (environment variable)

เป็นตัวแปรที่ถูกกำหนดขึ้นจากคำสั่งในเชลล์ (shell) เช่น

```
setenv PATH /bin:/etc:
```

ทำให้โปรเซสสามารถดึงข้อมูลเหล่านี้มาใช้โดยใช้ฟังก์ชั่น getenv() ตัวแปรที่เกี่ยวข้องมีดังนี้

THIN อาจมีค่าเป็น 1 หรือ 2 หรือ 3 หรือ 4
ค่าของตัวแปร THIN จะหมายถึงลักษณะการจัดระดับตัวอักษรในขณะที่ทำการป้อนข้อมูลเข้า ซึ่งจะสัมพันธ์กับรูทีน thread() เช่น 1 จะ แสดงตัวอักขระต่างๆใน 1 บรรทัด

THOUT อาจมีค่าเป็น 1 หรือ 2 หรือ 3 หรือ 4 หรือ P
ค่าของตัวแปร THOUT จะหมายถึงลักษณะการจัดระดับตัวอักษรในขณะที่สั่งแสดงผลบนจอภาพ ซึ่งสัมพันธ์กับรูทีน thwrite() สำหรับแบบ P นั้น จะแสดงแบบ 3 ระดับ โดยแยกออกเป็น 3 บรรทัดก่อนที่จะแสดงออกไป

THCODE อาจมีค่าเป็น 7 หรือ 8
ค่าของตัวแปร THCODE จะหมายถึงจำนวนบิตของข้อมูลแต่ละตัวอักษรที่ถูกป้อนเข้ามาจากแป้นพิมพ์ ถ้าหากเป็น 8 หมายถึงข้อมูลจากเทอร์มินัลอาจจะเป็นภาษาไทยก็ได้ เช่น เทอร์มินัล V382 หรือไมโครคอมพิวเตอร์ที่ต่อเข้ากับระบบผ่านโปรแกรมตัวเลียน (program emulator) ซึ่งสามารถส่งข้อมูลทั้ง 8 บิตให้กับโปรแกรมย่อยควบคุมภาษาไทย

THMODE อาจมีค่าเป็น RAW หรือ CANON
หมายถึงลักษณะการส่งข้อมูลให้แก่โปรแกรมของผู้ใช้เป็นแบบรอว์หรือคานอน

ในกรณีที่ตัวแปรเหล่านี้ไม่มีการกำหนดมาจากคำสั่งในเชลล์ โปรแกรมย่อยควบคุมภาษาไทย จะกำหนดค่าโดยปริยาย (default) ดังนี้

THIN มีค่าเป็น 3
THOUT " P
THCODE " 7
THMODE " CANON

ค่าของตัวแปรอาจเปลี่ยนแปลงได้จากรูทีน thioctl() จากโปรแกรมของผู้ใช้เอง ซึ่งจะกล่าวต่อไปในภายหลัง

2. แฟ้มข้อมูล /etc/termcap

เนื่องมาจากความแตกต่างของเทอร์มินัลจากบริษัทผู้ผลิตต่างๆ ทำให้เกิดปัญหาต่อโปรแกรมอย่างเช่น โปรแกรมบรรณาธิการ vi ที่ต้องจัดการกับข้อความต่างๆ เช่น การเลื่อนหน้า การโยกย้ายตำแหน่งของเคอร์เซอร์ การลบบรรทัด การแทรกตัวอักษร ซึ่งลักษณะเหล่านี้ขึ้นกับฮาร์ดแวร์ กับ เฟิร์มแวร์ที่มีอยู่ในแต่ละเทอร์มินัล แต่โปรแกรม vi จำต้องสามารถทำงานได้บนทุกๆเทอร์มินัล ซึ่งมีการใช้คอนโทรลโคดที่แตกต่างกัน ดังนั้นแทนที่จะฝังคอนโทรลโคดเหล่านั้นลงในโปรแกรม vi ก็ทำการแยกคอนโทรลโคดต่างๆ ออกเป็นแฟ้มข้อมูลคือ /etc/termcap และกำหนดให้มีฟังก์ชั่นต่างๆ เพื่อดึงเอาคอนโทรลโคดออกมาใช้

สำหรับโปรแกรมย่อยควบคุมภาษาไทยก็เช่นเดียวกับโปรแกรม vi จำเป็นต้องใช้คอนโทรลโคดในการแสดงอักขระภาษาไทยในระดับต่างๆ จึงต้องดึงข้อมูลที่มีอยู่ใน termcap มาใช้ ก่อนที่โปรแกรมของผู้ใช้จะนำเอาข้อมูลที่มีอยู่ใน termcap มาใช้ จำต้องรู้ว่าเทอร์มินัลที่ใช้อยู่ชื่ออะไร โดยผ่านตัวแปรเอนไวรอนเมนท์ TERM เพื่อใช้หาเอนทรีของเทอร์มินัลนั้นใน termcap ทั้งนี้เนื่องจากข้อมูลที่อยู่ใน termcap สำหรับแต่ละเทอร์มินัลประกอบด้วยกัน 2 ส่วนด้วยกันคือ ส่วนที่เป็นชื่อของเทอร์มินัลหรือนามแฝงของเทอร์มินัลที่อาจจะมีหลายชื่อก็ได้ แต่จะมีเพียงชื่อเดียวเท่านั้นที่ตรงกับค่าของตัวแปร TERM ส่วนที่สองจะแสดงคอนโทรลโคดต่างๆซึ่งเขียนขึ้นภายใต้รูปแบบที่กำหนดไว้ สำหรับเทอร์มินัลวีที 220 ค่าของ TERM จะมีค่าเป็น vt100

ตัวอย่างที่ 3.1

```
1.  vt100|vt100-am|dec vt100:\
2.    :cr=^M:nl=^J:bl=^G:cl=\E[;H\E[2J:\
3.    :le=^H:up=\E[A:do=^J
```

บรรทัดที่ 1 เป็นชื่อและนามแฝงของเทอร์มินัล ซึ่งค่าของตัวแปร TERM เมื่อมีค่าเป็น vt100 หรือ vt100-am หรือ dec vt100 ต่างก็จะหมายถึงเอนทรีเดียวกันใน termcap

บรรทัดที่ 2 เป็นบรรทัดที่แสดงความสามารถของเทอร์มินัล เช่น cl หมายถึงการล้างจอภาพ จะต้องใช้เอสเคปซีเควนซ์ ESC [; H ESC [2 J (\E หมายถึง ESC) ซึ่งโปรแกรมย่อยควบคุมภาษาไทยเรียกใช้เฉพาะส่วนที่เลื่อนตำแหน่งของเคอร์เซอร์ คือ le เลื่อนเคอร์เซอร์ไปทางซ้าย 1 ตำแหน่ง up เลื่อนเคอร์เซอร์ไปข้างบน 1 ตำแหน่ง และ do เลื่อนเคอร์เซอร์ลงมา 1 ตำแหน่ง

3. การควบคุมอุปกรณ์อินพุทเอาท์พุท

ในบางครั้งงานบางงานก็ต้องการให้โปรแกรมย่อยควบคุมเทอร์มินัลทำงานแตกต่างไปจากสภาวะเดิมจากที่เป็นอยู่ เช่น ในการป้อนรหัสลับ (password) นั้นไม่ต้องการให้ผู้อื่นเห็นรหัสลับที่ป้อนเข้าไป ก็สามารถกำหนดให้โปรแกรมย่อยควบคุมเทอร์มินัลไม่ทำการแสดงผลข้อมูลที่ถูกป้อนเข้าไป หรือ การที่ผู้ใช้ต้องการเปลี่ยนแป้นอักษรสำหรับคอนโทรลคาแรคเตอร์ เช่น erase หรือ kill หรือ interrupt

ระบบปฏิบัติการยูนิกซ์ได้กำหนดให้ system call ชื่อ ioctl ทำหน้าที่ในการควบคุมอุปกรณ์อินพุทเอาท์พุทต่างๆ ซึ่งประกอบด้วยพารามิเตอร์ 3 ตัวด้วยกัน ioctl(fd, cmd, arg) fd คือ หมายเลขของแฟ้ม cmd คือ คำสั่งที่ต้องการให้ ioctl ทำ ส่วน arg ก็เป็น argument ที่ cmd ต้องการใช้ ส่วนใหญ่จะเป็น pointer ชี้ไปยังข้อมูลที่เกี่ยวข้องกับเทอร์มินัลซึ่งผู้ใช้สามารถเปลี่ยนแปลงได้ มีอยู่ด้วยกัน 3 ชุด

sgttyb structure

```
struct sgttyb {
    char sg_ispeed; /* input speed */
    char sg_ospeed; /* output speed */
    char sg_erasec; /* erase character */
    char sg_kill    /* kill character */
    short sg_flags   /* mode flags */
}
```

sg_ispeed และ sg_ospeed ใช้ในการกำหนดความเร็วในการรับและส่งข้อมูลกับเทอร์มินัล ซึ่งอาจจะมีค่าต่างๆดังนี้ B0, B50, B75, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600 sg_erasec เป็นตัวอักษรบนแป้นพิมพ์เพื่อใช้ในการลบตัวอักษรที่อยู่ก่อนหน้า sg_kill เป็นตัวอักษรบนแป้นพิมพ์เพื่อใช้ในการลบตัวอักษรทั้งบรรทัด ซึ่งโปรแกรมย่อยควบคุมภาษาไทยใช้ sg_erasec และ sg_kill เปรียบเทียบตัวอักษรแต่ละตัวที่ถูกป้อนเข้ามาว่าเป็นตัวอักขระนิเศษหรือไม่ ถ้าหากใช่ก็แสดงการลบตัวอักษรบนจอภาพ และในบัฟเฟอร์ ส่วน sg_flags ประกอบด้วย 16 บิต ซึ่งแต่ละบิตก็จะแทนโหมดต่างๆของเทอร์มินัล โดยโปรแกรมย่อยควบคุมภาษาไทยมีการ เรียกใช้เพียง 2 บิตเท่านั้น คือ

ECHO — ถ้าหากทำการเซ็ท ระบบปฏิบัติการจะแสดงตัวอักษรที่ป้อนเข้ามาจากแป้นพิมพ์ออกทางจอภาพถ้าหากไม่เซ็ทก็ไม่แสดงออกทางจอภาพ เคอร์เซอร์ก็ไม่เคลื่อนที่

RAW — ถ้าหากทำการเซ็ทจะไม่ทำการประมวลผลข้อมูลทำให้ข้อมูลมีครบทั้ง 8 บิต เมื่อได้รับข้อมูลจะส่งให้กับโปรแกรมของผู้ใช้ทันที จะไม่เก็บข้อมูลลงในบัฟเฟอร์เพื่อรอรีเทอร์น นอกจากนี้ตัวอักษรนิเศษ ที่ใช้ลบตัวอักษร หรืออินเทอร์รัพท์จะไม่ถูกตีความ แต่จะถูกมองเป็นข้อมูลธรรมดาเท่านั้น

tchar structure

```
struct tchars {
        char t_intrc;  /* interrupt default ^C*/
        char t_quitc;  /* quit default ^\*/
        char t_startc; /* start output defautl ^Q*/
        char t_stopc;  /* stop output default ^S*/
        char t_eofc;   /* end of file default ^D*/
        char t_brkc;   /* input delimeter */
    }
```

โครงสร้างแบบที่สองนี้ใช้ในการกำหนดตัวอักษรจากแป้นพิมพ์สำหรับแต่ละเทอร์มินัล t_intrc เป็นตัวอักขระเพื่อใช้ในการยกเลิกโปรเซสในขณะที่ทำงานอยู่ t_quitc เป็นตัวอักขระเพื่อใช้ในการยกเลิกโปรเซสพร้อมทั้งบันทึกการทำงานของโปรเซสในขณะนั้นลงในแฟ้มข้อมูลชื่อ core t_stopc ใช้เพื่อหยุดการแสดงผลบนจอภาพชั่วคราวจนกว่าจะเจอ t_startc จึงจะแสดงผลต่อ t_eofc ใช้เพื่อบอกจุดสิ้นสุดของข้อมูล

สำหรับโปรแกรมย่อยควบคุมภาษาไทยจะใช้เฉพาะ t_intr t_quitc t_stopc t_startc และ t_eofc พร้อมทั้งเลียนแบบการทำงานของโปรแกรมย่อยควบคุมเทอร์มินัลเมื่อพบตัวอักขรพิเศษเหล่านี้โดยถ้าเป็น t_intr โปรแกรมย่อยควบคุมภาษาไทยจะยกเลิกโปรเซสที่กำลังทำงานอยู่ t_quitc ก็ใช้ยกเลิกโปรเซสพร้อมสร้างแฟ้มข้อมูล core ส่วน t_stopc และ t_startc โปรแกรมย่อยควบคุมภาษาไทยใช้ควบคุมการไหลของข้อมูลในการแสดงผลออกทางจอภาพ และ t_eofc ใช้ในการบอกจุดสิ้นสุดของข้อมูล

ltchar structure

```
struct ltchar {
        char t_suspc;  /* stop process signal */
        char t_dsuspc; /* delay stop process signal */
        char t_rprntc; /* reprint line */
        char t_flushc; /* flush output */
        char t_werase; /* word erase */
        char t_lnextc; /* literal next character */
    }
```

สำหรับโครงสร้าง ltchar นี้จะใช้ในการกำหนดตัวอักษรจากแป้นพิมพ์สำหรับแต่ละเทอร์มินัล t_suspc ใช้เพื่อหยุดการทำงานของโปรเซสชั่วคราว t_rprntc ใช้เพื่อพิมพ์ข้อความในบรรทัดนั้นใหม่ t_werase ใช้เพื่อลบคำที่อยู่ก่อน 1 คำ t_lnextc ใช้เพื่อต้องการป้อนตัวอักษรที่เป็นคอนโทรลคาแรคเตอร์ ในการใช้ต้องป้อนตัวอักษร t_lnextc ก่อนแล้วตามด้วยคอนโทรลคาแรคเตอร์ที่ต้องการ

สำหรับโครงสร้างนี้โปรแกรมย่อยควบคุมภาษาไทยมีการเลียนแบบการทำงานเพียงตัวอักขระเดียวเท่านั้นคือ t_suspc คือหยุดการทำงานของโปรเซสชั่วคราว

นอกจากจะใช้ system call ioctl ในการเปลี่ยนแปลงค่าต่างๆ จากในโปรแกรมแล้ว ในส่วนที่เป็น shell อาจใช้คำสั่ง stty เพื่อเปลี่ยนค่าต่างๆให้มีลักษณะตามที่ต้องการได้เช่นกัน โดยคำสั่ง stty จะทำการเรียก system call ioctl() อีกต่อหนึ่ง

4. ซิกแนล

ซิกแนลนั้นถือได้ว่าเป็นการขัดจังหวะโดยซอฟต์แวร์ที่สามารถส่งให้กับโปรเซสใดๆ เพื่อบอกให้รู้ถึงสิ่งที่เกิดขึ้น บางซิกแนลเช่น floating point exception เป็นส่วนที่เกิดขึ้นจากฮาร์ดแวร์โดยตรง หรือ ซิกแนลที่บอกถึงการเปลี่ยนสถานะของโปรเซสลูกเกิดจากซอฟต์แวร์ ซิกแนลมาตรฐานโดยส่วนใหญ่จะมีผลให้โปรเซสเลิกการทำงาน แต่บางโปรเซสก็มีการบันทึกการทำงานของโปรเซสลงในแฟ้มข้อมูลชื่อ core เมื่อได้รับซิกแนล ซึ่งจะมีประโยชน์ในการค้นหาที่ผิดของโปรแกรม

ในบางครั้งขณะที่ทำคำสั่งในยูนิกซ์ เช่น คอมไพล์โปรแกรม

```
$ cc more.c
```

เมื่อผู้ใช้ต้องการยกเลิกการคอมไพล์โดยการกด ctrl-C สิ่งที่เกิดขึ้นก็คือ ส่วนของเคอร์เนลที่ดูแลข้อมูลจากแป้นพิมพ์พบว่าข้อมูลที่เข้ามาเป็นอินเทอร์รัพท์ ก็จะส่งซิกแนลชื่อว่า SIGINT ไปยังทุกโปรเซสที่อยู่ภายใต้ตัวควบคุมเทอร์มินัลเดียวกันเมื่อโปรเซส cc ได้รับซิกแนลก็จะทำพฤติกรรมโดยปริยาย (default action) แล้วหยุดการทำงาน แต่สำหรับเชลล์เมื่อได้รับซิกแนลกลับยังทำงานต่อไปได้ ทั้งนี้เชลล์นั้นไม่สนใจต่อซิกแนลที่ส่งมา นั่นคือโปรแกรมสามารถเลือกที่จะจับซิกแนลหรือไม่สนใจซิกแนลก็ได้

นอกจากการส่งซิกแนลจากเคอร์เนลไปยังโปรเซสแล้ว ยังสามารถส่งจากโปรเซสหนึ่งไปอีกโปรเซสหนึ่งได้เช่น ถ้าหากต้องการคอมไพล์โปรแกรมใน background

```
$ cc more.c &
  1098
```

1098 หมายถึงหมายเลขของโปรเซสของ cc ที่ส่งไปทำงานใน background การสั่งยกเลิกโปรเซส cc ทำได้โดยใช้คำสั่ง kill ตามด้วยหมายเลขของโปรเซส เช่น

```
$ kill 1098
```

จากตัวอย่างจะเป็นการส่งซิกแนลที่ชื่อ SIGTERM จากโปรเซสเชลล์ไปยังโปรเซสหมายเลข 1098 ซึ่ง SIGTERM จะมีผลทำให้ โปรเซส cc หยุดการทำงานเช่นเดียวกับ SIGINT

สำหรับซิกแนลต่างๆที่มีใช้อยู่ในยูนิกซ์จะแตกต่างกันไปตามแต่ละตระกูล แต่ซิกแนล 15 ตัวแรกนั้น ถือได้ว่าเป็นซิกแนลมาตรฐานที่ทุกตระกูลต้องมี สำหรับตระกูลบีเอสดี 4.2 นั้นมีทั้งหมด 27 ซิกแนล และมีการเพิ่มเติมอีกจนเป็น 30 ซิกแนลในบีเอสดี 4.3 แต่ในที่นี้จะขออธิบายถึงซิกแนลเฉพาะที่ใช้ในโปรแกรมย่อยควบคุมภาษาไทย

SIGINT	(2)	ซิกแนลนี้จะถูกส่งจากเคอร์เนลไปยังทุกโปรเซสซึ่งเกี่ยวข้องกับเทอร์มินัลที่ผู้ใช้กดปุ่มอินเทอร์รัพท์ เพื่อหยุดการทำงานของโปรแกรม เกิดขึ้นได้ถ้าผู้ใช้กด ctrl-C
SIGQUIT	(3)	SIGQUIT จะคล้ายกับ SIGINT แต่จะสร้างแฟ้มข้อมูล

ชื่อ core เพื่อใช้หาที่ผิดของโปรแกม เกิดขึ้นได้ถ้าผู้ใช้กด ctrl-\

SIGTSTP (18) ซิกแนลนี้ใช้เพื่อหยุดโปรเซสชั่วคราว เกิดขึ้นได้ถ้าผู้ใช้กด ctrl-Z

SIGCONT (19) ซิกแนลนี้ใช้เพื่อต้องการให้โปรเซสที่หยุดทำงานต่อ

SIGIO (23) ข้อมูลสามารถอ่านหรือส่งไปให้แฟ้มข้อมูล(ที่เป็นเทอร์มินัล) ได้ทันทีโดยไม่ต้องรอ

การจัดการกับซิกแนล

ในการเขียนโปรแกรมเพื่อจัดการกับซิกแนลต่างๆนั้น ต้องเรียกผ่าน system call signal ซึ่งประกอบด้วย argument 2 ตัวด้วยกัน argument ตัวแรกจะระบุถึงชื่อของซิกแนล ที่ได้กำหนดไว้ในแฟ้มข้อมูลชื่อ signal.h ส่วน argument ตัวที่ 2 อาจจะมีค่าเป็น SIG_DFL ซึ่งหมายถึงให้กระทำแบบปริยาย หรือ SIG_IGN หมายถึงไม่ต้องสนใจซิกแนลที่ระบุไว้ใน argument 1 หรือ ชื่อของรูทีนใดๆที่ต้องการให้ทำเมื่อได้รับซิกแนล

การละเลยซิกแนล

การละเลยซิกแนลทำได้โดย

```
signal(signame, SIG_IGN)
```

การกำหนดเช่นนี้มีผลให้ไม่สนใจต่อซิกแนลเมื่อได้รับจนกว่าจะมีการกำหนดการจัดการกับซิกแนลนั้นใหม่

ตัวอย่างที่ 3.1

```
#include <signal.h>
main()
{
    signal(SIGINT, SIG_IGN);
    pause();
}
```

จากตัวอย่างที่ 3.1 การยกเลิกโปรแกรมโดยการส่งซิกแนล SIGINT นั้นไม่สามารถทำได้ เพราะได้กำหนดให้ละเลยต่อ SIGINT แต่การยกเลิกโปรแกรมอาจทำได้โดยส่งซิกแนลอื่นเช่น SIGQUIT ที่มีผลให้โปรแกรมหยุดทำงานได้เช่นกัน หรือ SIGKILL ซึ่งเป็นซิกแนลที่กำหนดให้ไม่สามารถถูกละเลยได้

การดักจับซิกแนล

การเขียนโปรแกรมเพื่อดักจับซิกแนลได้แสดงดังตัวอย่าง 3.2

ตัวอย่างที่ 3.2

```
#include <signal.h>
main()
{
     extern int handler();
     signal(SIGINT, handler);
     for(;;)
          pause();
}
handler()
{
     printf("OUCH\n");
}
```

จากตัวอย่างที่ 3.2 เมื่อได้รับซิกแนล SIGINT โปรแกรมจะพิมพ์ OUCH สำหรับยูนิกซ์ที่ไม่ใช่ตระกูลบีเอสดี เมื่อได้รับซิกแนล SIGINT ครั้งที่สองจะมีผลให้โปรแกรมหยุดการทำงาน เพราะการกำหนดรูทีนที่ทำงานเมื่อได้รับซิกแนลจะมีผลเพียงครั้งเดียว ดังนั้นเมื่อได้รับ SIGINT ครั้งที่สองรูทีนที่ทำงานก็จะกลายเป็น SIG_DFL ไปซึ่งมีผลให้โปรแกรมหยุดการทำงาน ในตัวอย่างที่ 3.3 จึงเพิ่มการจับซิกแนลใน handler อีก

ตัวอย่างที่ 3.3

```
#include <signal.h>
main()
{
     extern int handler;
     signal(SIGINT, handler);
     for(;;)
          pause();
}
handler()
{
     printf("OUCH\n);
     signal(SIGINT, handler);
}
```

จากตัวอย่างที่ 3.3 จะมีผลให้โปรแกรมนิมน์ OUCH ทุกครั้งที่ผู้ใช้กดปุ่มอินเทอร์รัพท์ แต่สำหรับยูนิกซ์ตระกูลบีเอสดีตัวอย่างที่ 3.2 และ 3.3 จะให้ผลเหมือนกัน แต่ปัญหาที่เกิดขึ้นก็คือเมื่อเกิดกดปุ่มอินเทอร์รัพท์ในระหว่างที่ทำงาน handler นั้น สำหรับในตระกูลบีเอสดีจะทำให้มีการเรียก handler ซ้อนกันแต่ถ้าหาก handler นั้นมีการทำงานที่ค่อนข้างสั้น โอกาสที่จะเกิดซิกแนลอินเทอร์รัพท์ใน handler ก็น้อย วิธีการแก้ไขทำได้โดยเพิ่มคำสั่งการเรียก signal ใน handler รูทีนเป็นคำสั่งแรก และให้ argument ตัวที่ 2 เป็น SIG_IGN ดังนี้ เพื่อให้ละเลยต่อซิกแนลที่ได้รับขณะที่ทำงานในรูทีน handler

ตัวอย่างที่ 3.4

```
#include <signal.h>
main()
{
     extern int handler;
     signal(SIGINT, handler);
     for(;;)
          pause();
}
```

```
handler()
{
     signal(SIGINT, SIG_IGN);
     printf("OUCH\n");
     signal(SIGINT, handler);
}
```

จากหลักการทำงานของซิกแนลที่ได้กล่าวมาแล้วข้างต้น โปรแกรมย่อยควบคุมภาษาไทยได้ใช้หลักการเหล่านี้ช่วยในการอ่านข้อมูลที่ถูกป้อนจากแป้นพิมพ์และควบคุมการไหลของข้อมูลโดยอาศัยสัญญาณ SIGIO และใช้สัญญาณ SIGINT SIGQUIT SIGTSTP SIGCONT ควบคุมการทำงานของโปรเซส