บทที่ 3
การออกแบบและพัฒนาโปรแกรม

3.1 เทคนิคในการพัฒนาโปรแกรมย่อยควบคุมอุปกรณ์

3.1.1 ข้อแตกต่างของการพัฒนาโปรแกรมย่อยควบคุมอุปกรณ์ (device driver) และโปรแกรมประยุกต์ (application program) จะขอกล่าวถึงความแตกต่างเป็นหัวข้อ ดังนี้

3.1.1.1 แฟ้มผลลัพธ์จากการแปลด้วยตัวแปลชุดคำสั่ง (output file)

โปรแกรมประยุกต์เมื่อผ่านการแปลด้วยตัวแปลชุดคำสั่งจะได้ แฟ้มโปรแกรมที่ทำงานได้ (executable) ซึ่งจะเป็นชื่อ a.out โดยปริยาย เมื่อเรียกให้โปรแกรมทำงาน โปรแกรมจะทำงานภายในขอบเขตของหน่วยความจำหลักช่วงหนึ่ง และภายใต้การดูแลของเคอร์เนล

โปรแกรมควบคุมอุปกรณ์ การแปลจะแปลให้เป็นแฟ้มรหัสจุดหมาย (object code) เท่านั้น โปรแกรมจะทำงานได้ต้องผ่านของการเชื่อมโยง (link) เข้ากับรหัสจุดหมายทั้งหมดของเคอร์เนล เพื่อให้เกิดโปรแกรมที่ทำงานได้เพียงโปรแกรมเดียว ซึ่งก็คือเคอร์เนลใหม่ ดังนั้นโปรแกรมควบคุมอุปกรณ์เป็นเพียงส่วนหนึ่งของโปรแกรมที่ทำงานได้เท่านั้น

3.1.1.2 รูปแบบของโปรแกรม (format)

โปรแกรมประยุกต์จะประกอบด้วยโปรแกรมหลัก (main routine) หนึ่งโปรแกรม และโปรแกรมย่อยหรือฟังก์ชั่นมากมาย

โปรแกรมควบคุมอุปกรณ์ไม่มีโปรแกรมหลัก มีแต่ฟังก์ชั่นในการทำงานติดต่อกับอุปกรณ์ต่างๆ โดยตัวเคอร์เนลทำหน้าที่เป็นโปรแกรมหลัก

3.1.1.3 การทำงานของโปรแกรม (Execution)

โปรแกรมประยุกต์จะถูกเรียกขึ้นมาทำงานโดยโปรเซสเดียว ถ้ามีหลายโปรเซสเรียกใช้โปรแกรมประยุกต์ เคอร์เนลจะอ่านโปรแกรมขึ้นมาหลายชุด แต่ละโปรเซสทำงานกับโปรแกรมแต่ละชุด (ยกเว้นเมื่อมีการ fork() โปรแกรมใหม่)

โปรแกรมควบคุมอุปกรณ์เป็นส่วนหนึ่งของเคอร์เนล โดยปกติการทำงานของเคอร์เนลไม่ว่าจะมีโปรเซสเรียกใช้งานมากเท่าไร จะมีเคอร์เนลเพียงชุดเดียวที่รองรับการใช้งาน ดังนั้นโปรแกรมควบคุมอุปกรณ์ต้องพัฒนาในรูปแบบที่จะต้องสามารถรองรับการเรียกใช้งานจากหลายโปรเซสได้ในเวลาเดียวกัน

3.1.1.4 การทำงานของโปรแกรมย่อย (Routine execution)

โปรแกรมย่อยของโปรแกรมประยุกต์จะทำงานตามลำดับคำสั่งที่เรียกใช้จากโปรแกรมหลัก (main) หรือจากการเรียกใช้ของโปรแกรมย่อยอื่น

โปรแกรมย่อยของโปรแกรมควบคุมอุปกรณ์ จะทำงานก็ต่อเมื่อโปรเซสเรียกใช้ฟังก์ชั่นระบบ (system call) ดังนั้นการทำงานของโปรแกรมย่อยของโปรแกรมควบคุมอุปกรณ์แต่ละโปรแกรมจะไม่มีความสัมพันธ์กัน และไม่ได้เกิดขึ้นตามลำดับขั้นตอน

3.1.1.5 เวลาที่ใช้ในการทำงาน (Timing)

โปรแกรมประยุกต์ทำงานตามคำสั่ง หรือเรียกใช้ฟังก์ชั่นระบบ (system call) โดยไม่ต้องคำนึงถึงว่าจะต้องใช้เวลาในการทำงานนานเท่าไร หรือมีสัญญาณขัดจังหวะจากอุปกรณ์หรือไม่

โปรแกรมควบคุมอุปกรณ์ การทำงานของโปรแกรมย่อยในเคอร์เนลแต่ละโปรแกรมจะมีระยะเวลาในการทำงานที่แตกต่างกัน ระยะเวลาในการทำงานของแต่ละคำสั่งจะมีผลต่อการทำงานของโปรแกรมควบคุมอุปกรณ์โดยรวมกล่าวคือ การทำงานของโปรแกรมควบคุมอุปกรณ์อาจเปลี่ยนแปลงได้ ถ้าช่วงระยะเวลาในการทำงานเปลี่ยนแปลงไป คำสั่งที่มีผลต่อระยะเวลาในการทำงาน คือคำสั่งที่ใช้ในการเขียนข้อความบนจอภาพ เช่น cmn_err(), print() เป็นต้น

3.1.1.6 การขัดจังหวะ (Interrupts)

โปรแกรมประยุกต์ไม่ต้องคำนึงถึงการขัดจังหวะของอุปกรณ์ เนื่องจากจะไม่มีผลต่อการทำงานของโปรแกรมเลย

โปรแกรมควบคุมอุปกรณ์ที่ควบคุมอุปกรณ์จริง (real device) จะต้องพัฒนาโปรแกรมให้มีกลไกอย่างน้อย 2 อย่างที่เกี่ยวข้องกับการขัดจังหวะ คือ

- ต้องมีโปรแกรมตอบรับสัญญาณขัดจังหวะที่เกิดขึ้น และดำเนินการตามความเหมาะสม
- ต้องเขียนโปรแกรมให้มีการป้องกันความผิดพลาดที่จะเกิดขึ้นกับบางส่วนของโปรแกรมเนื่องจากสัญญาณขัดจังหวะ เช่น ในขั้นตอนการจัดการกับรายการเชื่อมโยง (linked list) ส่วนของโปรแกรมที่ต้องได้รับการป้องกันจากการขัดจังหวะ จะเรียกว่า critical code section

การป้องกันการขัดจังหวะทำได้โดยการเพิ่มระดับความสำคัญ (priority level) ของเคอร์เนลในขณะนั้นขึ้น ซึ่งจะทำให้สัญญาณขัดจังหวะที่มีระดับความสำคัญต่ำกว่าระดับความสำคัญของเคอร์เนลในขณะนั้นไม่สามารถขัดจังหวะการทำงานได้

3.1.1.7 ความสามารถที่จะถูกเรียกให้ทำงานกระบวนงานใหม่ก่อนสิ้นสุดการทำงานตามกระบวนงานเดิม (re-entrant processing)

สำหรับระบบปฏิบัติงานในลักษณะหลายภาระกิจ (multitasking) ในขณะหนึ่งจะมีหลายโปรเซสที่ทำงานอยู่ และอาจจะเป็นการทำงานในส่วนเดียวกันของเคอร์เนลด้วย โปรแกรมควบคุมอุปกรณ์ต้องพัฒนาให้สามารถรองรับการถูกเรียกใช้งานจากหลายโปรเซสโดยไม่จำเป็นว่าการทำงานครั้งที่แล้วจะต้องเสร็จสิ้นก่อนที่จะมีการเรียกใช้งานครั้งต่อไป

3.1.2 การพัฒนาโปรแกรมเพื่อให้เกิดการประสานจังหวะ ภายในโปรแกรมควบคุมอุปกรณ์ (Synchronization within the driver)

การพัฒนาโปรแกรมควบคุมอุปกรณ์ต้องระมัดระวังในการพัฒนา เพื่อให้เกิดการประสานจังหวะที่ดีระหว่าง

- โปรแกรมควบคุมอุปกรณ์และอุปกรณ์
- โปรแกรมย่อยควบคุมอุปกรณ์หนึ่งกับโปรแกรมย่อยควบคุมอุปกรณ์อื่นๆ ของโปรแกรมควบคุมอุปกรณ์เดียวกัน
- การถูกเรียกใช้งานหลาย ๆ ครั้งในเวลาเดียวกันหรือเวลาใกล้เคียงกันของโปรแกรมย่อยควบคุมอุปกรณ์เดียวกัน

เคอร์เนลของยูนิกซ์ได้เตรียมกลไกต่าง ๆ ไว้เพื่อให้โปรแกรมควบคุมอุปกรณ์ใช้ในการประสานจังหวะกัน ดังนี้

1. การใช้เคอร์เนลฟังก์ชั่น sleep() และ wakeup()

โปรแกรมควบคุมอุปกรณ์สามารถใช้เคอร์เนลฟังก์ชั่น sleep() และ wakeup() ในกรณีต่อไปนี้ คือ

- โปรแกรมควบคุมอุปกรณ์ได้สั่งคำสั่งให้อุปกรณ์ทำงาน และจะต้องรอจนกระทั่งการทำงานตามคำสั่งนั้นของอุปกรณ์เสร็จสิ้นลง
- โปรแกรมควบคุมอุปกรณ์ต้องการจะเข้าใช้ทรัพยากรบางอย่างที่กำลังจะถูกใช้งานอยู่โดยโปรเซสอื่น ดังนั้นจึงต้องรอจนกว่าทรัพยากรนั้นว่างให้ใช้งาน

เมื่อโปรแกรมควบคุมอุปกรณ์เรียกใช้ฟังก์ชั่น sleep() เคอร์เนลจะหยุดการทำงานของโปรแกรมควบคุมอุปกรณ์นั้นไว้ รอจนกว่าจะมีการเรียกใช้ฟังก์ชั่น wakeup() ที่สัมพันธ์กันเคอร์เนลจึงจะเรียกโปรแกรมควบคุมอุปกรณ์นี้มาทำงานต่อ

การเรียกใช้ฟังก์ชั่น sleep() ต้องระบุพารามิเตอร์เป็นช่องทางสัญญาณที่รอ(channel) ซึ่งจะเป็นพารามิเตอร์เดียวกันกับที่โปรแกรมอื่นต้องใช้เพื่อ wakeup() โปรแกรมที่ sleep() นี้ช่องทางสัญญาณที่เป็นพารามิเตอร์นี้ จะเป็นตัวเลขอะไรก็ได้ที่ทั้งโปรแกรมควบคุมอุปกรณ์ที่เรียกฟังก์ชั่น sleep() และโปรแกรมที่เรียกฟังก์ชั่น wakeup() รับรู้ด้วยกัน โดยปกติ

มักจะใช้เลขที่อยู่ (address) ของโครงสร้างข้อมูลสถิติที่เกี่ยวข้องกับเหตุการณ์ที่รอ เช่น ขณะที่โปรแกรมส่งโครงสร้างข้อมูลร้องขอไปยังโปรแกรมควบคุมสกาสซี่โฮสอแคปเตอร์ และต้องรอจนกว่าอุปกรณ์เป้าหมายทำงานเสร็จ จะเรียกใช้ฟังก์ชั่นระบบ sleep() โดยใช้ช่องทางสัญญาณ คือ ตำแหน่งที่อยู่ของโครงสร้างข้อมูลร้องขอ ซึ่งเมื่ออุปกรณ์เป้าหมายทำงานเสร็จ จะส่งสัญญาณขัดจังหวะให้โปรแกรมตอบรับการขัดจังหวะขึ้นมาทำงาน โดยส่งพารามิเตอร์เป็นโครงสร้างข้อมูลร้องขอ โปรแกรมตอบรับการขัดจังหวะก็จะใช้ตำแหน่งที่อยู่ของโครงสร้างข้อมูลร้องขอเป็นพารามิเตอร์ของฟังก์ชั่น wakeup() เพื่อปลุกโปรแกรมที่ sleep() ได้

อาจจะมีหลายโปรเซสที่ sleep() โดยใช้ช่องทางสัญญาณเดียวกัน ซึ่งอาจจะเป็นเพราะว่ารอใช้ทรัพยากรเดียวกัน ดังนั้นขณะที่ได้รับสัญญาณ wakeup() ทุกโปรเซสจะถูกเรียกขึ้นมาให้พร้อมที่จะทำงาน แต่เคอร์เนลจะเรียกเพียงโปรเซสเดียวขึ้นมาทำงาน โดยพิจารณาตามระดับความสำคัญ (Scheduling priority) โปรเซสที่ถูกเรียกขึ้นมาทำงานจะเข้าใช้ทรัพยากร โปรเซสอื่นที่เหลือจะต้องเรียกฟังก์ชั่น sleep() อีกครั้งเพื่อรอทรัพยากรให้ว่างอีก ดังนั้นโปรแกรมควบคุมอุปกรณ์ที่ใช้ฟังก์ชั่น sleep() หลังจากถูกปลุกขึ้นมาทำงานแล้วจะต้องตรวจสอบว่าเหตุการณ์ที่รอเกิดขึ้นแล้วหรือยัง หรือ ทรัพยากรที่รอคอยอยู่ว่างแล้วหรือยัง ถ้ายังก็มีความจำเป็นต้อง sleep() อีกครั้ง ตัวอย่างโปรแกรมคือ

ส่วนของโปรแกรมควบคุมอุปกรณ์

```
ptr_req->internal  = 1;
while (ptr_req->internal)
     sleep(ptr_req,PRIBIO);
```

ส่วนของโปรแกรมตอบรับสัญญาณขัดจังหวะ

```
ptr_req->internal  = 0;
wakeup(ptr_req);
```

สำหรับการหยุดรอเหตุการณ์ให้เกิดขึ้น โดยที่จะไม่มีขั้นตอนของสัญญาณขัดจังหวะเกิดขึ้นโปรแกรมควบคุมอุปกรณ์สามารถจะใช้ช่องทางสัญญาณที่ระบบปฏิบัติงานยูนิกซ์ เตรียมไว้คือ lbolt โดยเคอร์เนลจะเรียกฟังก์ชั่น wakeup() โดยใช้ช่องทางสัญญาณ lboltอัตโนมัติทุก ๆ วินาที ซึ่งโปรแกรมควบคุมอุปกรณ์จะถูกปลุกขึ้นมาตรวจสอบเหตุการณ์ได้อย่างสม่ำเสมอ

การใช้ฟังก์ชั่น sleep() ในโปรแกรมควบคุมอุปกรณ์ต้องแน่ใจว่า จะมีโปรแกรมอื่นมา wakeup() ได้ ไม่เช่นนั้นจะทำให้เกิดโปรเซสที่ sleep() ตลอดเวลา ดังนั้นจึงไม่ใช้ฟังก์ชั่น sleep() ในโปรแกรมตอบรับสัญญาณขัดจังหวะเนื่องจากจะไม่มีโปรแกรมอื่นมาปลุก

เคอร์เนลฟังก์ชั่นบางฟังก์ชั่น จะมีการเรียกใช้ฟังก์ชั่น sleep() และ wakeup() เช่น geteblk() จะเรียกฟังก์ชั่น sleep() ถ้าไม่มีบัฟเฟอร์ว่าง เป็นต้นและ

iodone() จะเรียกใช้ฟังก์ชั่น wakeup() เป็นต้น ดังนั้นต้องระวังไม่ใช้ฟังก์ชั่นระบบเหล่านี้ในส่วนของโปรแกรมย่อยที่ไม่มีโปรแกรมอื่นมาปลุก

2. ลำดับความสำคัญของการถูกเรียกทำงาน (scheduling priority)

การเรียกใช้เคอร์เนลฟังก์ชั่น sleep() จะต้องกำหนดลำดับความสำคัญของการถูกเรียกทำงาน เป็นพารามิเตอร์ที่สองด้วย ซึ่งลำดับความสำคัญของการถูกเรียกทำงานจะมีความหมาย 2 ประการ คือ

- เป็นค่าที่เคอร์เนลใช้ในการพิจารณาว่าจะจัดให้โปรเซสที่พร้อมจะทำงาน (executable proceses) โปรเซสใดขึ้นมาทำงาน โดยพิจารณาจากค่าลำดับความสำคัญนี้ ถ้าค่าลำดับความสำคัญเป็นตัวเลขน้อย จะมีความสำคัญมาก

- เป็นค่าบ่งชี้ว่าโปรเซสที่ sleep() นั้นจะถูกปลุกขึ้นมาทำงาน โดยสัญญาณ (signal) หรือไม่ โดยถ้าค่าลำดับความสำคัญของการถูกเรียกทำงานมีค่าน้อยกว่า หรือเท่ากับค่า PZERO (กำหนดไว้ในแฟ้มข้อมูล /usr/include/sys/param.h) โปรเซสจะไม่ถูกปลุกขึ้นมาทำงานโดยสัญญาณจากโปรแกรม (software interrupt) โปรเซสจะถูกปลุกขึ้นมาจากการเรียกใช้ฟังก์ชั่น wakeup() ในช่องทางสัญญาณเดียวกันกับที่ใช้ตอนเรียกฟังก์ชั่น sleep() เท่านั้น และเมื่อโปรเซสขึ้นมาทำงานแล้วจึงจะดำเนินการกับสัญญาณต่าง ๆ ที่ได้รับในช่วงที่ sleep()

โดยปกติแล้วการติดต่อกับอุปกรณ์ที่มีความเร็วในการทำงานสูง โปรแกรมจะให้อุปกรณ์ทำงานเสร็จก่อน แล้วจึงดำเนินการกับสัญญาณที่ได้รับ ดังนั้นจึงมักให้โปรแกรมเหล่านี้ sleep() โดยมีค่าลำดับความสำคัญของการถูกเรียกใช้คือ PRIBIO ซึ่งมีค่าต่ำกว่า PZERO เช่น โปรแกรมควบคุมอุปกรณ์แบบบล็อค สำหรับโปรแกรมที่ติดต่อกับอุปกรณ์ที่ทำงานช้า หรืออุปกรณ์ที่จะถูกพักการทำงานเป็นเวลานาน เช่น เทอร์มินัล หรือไปป์ มักจะให้โปรแกรมเหล่านี้ sleep() โดยมีค่าลำดับความสำคัญของการถูกเรียกใช้มีค่าสูงกว่า PZERO เช่นโปรแกรมควบคุมอุปกรณ์แบบคาแรคเตอร์ เช่น โปรแกรมควบคุมอุปกรณ์เทอร์มินัล

กำหนดลำดับความสำคัญของการถูกเรียกทำงานดังนี้

- สำหรับการรับข้อมูล PZERO + 5
- สำหรับการแสดงข้อมูล PZERO + 4

โปรแกรมควบคุมอุปกรณ์เครื่องพิมพ์

กำหนดความสำคัญของการถูกเรียกทำงาน PZERO + 5

3. ลำดับความสำคัญของอุปกรณ์และหน่วยประมวลผล (device and processor priority)

สัญญาณขัดจังหวะจากอุปกรณ์จะมีผลก่อให้เกิดการขัดจังหวะการทำงานขึ้น

ก็ต่อเมื่อลำดับความสำคัญการขัดจังหวะของอุปกรณ์มีค่ามากกว่าลำดับความสำคัญของหน่วยประมวลผล ถ้าไม่เช่นนั้นอุปกรณ์จะต้องรอให้ลำดับความสำคัญของหน่วยประมวลผลลดต่ำลงมาจนถึงระดับที่ต่ำกว่าลำดับความสำคัญของอุปกรณ์ก่อน จึงสามารถจะขัดจังหวะการทำงานได้

จากหลักการข้างต้น สามารถนำมาใช้ในการป้องกันการขัดจังหวะของอุปกรณ์ขณะที่มีการทำงานตามคำสั่งที่มีการเปลี่ยนแปลงโครงสร้างข้อมูลชุดเดียวกับที่ โปรแกรมตอบรับสัญญาณขัดจังหวะจะเปลี่ยนแปลง โดยการใช้คำสั่งเปลี่ยนลำดับความสำคัญของหน่วยประมวลผลให้มีลำดับความสำคัญเท่ากับหรือมากกว่าลำดับความสำคัญของการขัดจังหวะของอุปกรณ์ที่เกี่ยวข้อง หลังจากนั้นทำการเปลี่ยนแปลงโครงสร้างข้อมูลให้เสร็จ แล้วจึงปรับลำดับความสำคัญของหน่วยประมวลผลให้กลับคืนค่าเดิม เพื่อเปิดโอกาสให้มีการขัดจังหวะได้ต่อไป ตัวอย่างเช่น การจัดการกับรายการเชื่อมโยง (linked list) ของบัฟเฟอร์ในขั้นตอนการเรียงลำดับบัฟเฟอร์ของโปรแกรมย่อยกลยุทธ เป็นต้น

ตัวอย่างโปรแกรม

```
lev = spl5();
disksort(&Sromtab, bp);
splx();
```

โปรแกรมตอบรับการขัดจังหวะ ไม่ต้องเปลี่ยนลำดับความสำคัญของหน่วยประมวลผลเอง เนื่องจากเมื่อโปรแกรมตอบรับการขัดจังหวะทำงาน เคอร์เนลจะเปลี่ยนลำดับความสำคัญของหน่วยประมวลผลให้อยู่ที่ลำดับความสำคัญเดียวกับการขัดจังหวะของอุปกรณ์ เมื่อจบการทำงานของโปรแกรมตอบรับการขัดจังหวะ เคอร์เนลจะปรับลำดับความสำคัญของหน่วยประมวลผลตามเดิมโดยอัตโนมัติ เช่นเดียวกับเคอร์เนลฟังก์ชั่น geteblk() และ brelse() ซึ่งเกี่ยวข้องกับการเคลื่อนย้ายโครงสร้างข้อมูลของเคอร์เนลเข้า และออกจากรายการเชื่อมโยงจะเปลี่ยนลำดับ ความสำคัญของหน่วยประมวลผลให้อยู่ระดับ 6 หรือ 7 ซึ่งจะป้องกันการขัดจังหวะของอุปกรณ์ทุกชนิดและปรับค่าคืนสู่สภาวะเดิมเมื่อทำงานต่อเสร็จ

4. การควบคุมการถูกเรียกทำงานหลายครั้งของโปรแกรมควบคุมอุปกรณ์ (control of multiple executions of a driver)

ถ้าโปรแกรมย่อยควบคุมอุปกรณ์จะถูกเรียกใช้งานโดยโปรเซสหลายโปรเซสในเวลาใกล้เคียงกัน เช่น ในขณะที่โปรเซสหนึ่งเรียกโปรแกรมย่อยควบคุมอุปกรณ์เพื่ออ่านข้อมูลและเรียกฟังก์ชั่น sleep() เพื่อรอให้การอ่านข้อมูลของอุปกรณ์เสร็จ ในระหว่างนั้นอาจมีโปรเซสอื่นเรียกโปรแกรมย่อยควบคุมอุปกรณ์เดียวกันเพื่อทำการอ่านข้อมูลอีกบริเวณหนึ่ง ในลักษณะเช่นนี้โปรแกรมย่อยควบคุมอุปกรณ์ที่เรียกใช้งานทั้งสองครั้งนี้ ต่างก็ใช้เรจิสเตอร์ (register) และโครงสร้างข้อมูลเดียวกันในการทำงาน จะทำให้การทำงานของโปรแกรม

สับสนและเกิดความผิดพลาดขึ้น ดังนั้นจึงต้องมีระบบควบคุมให้เกิดการประสานจังหวะที่ดีของการทำงานหลาย ๆ ครั้งของโปรแกรมเดียวกัน

ระบบควบคุมสามารถสร้างได้โดยการกำหนดตัวบ่งชี้ (flag) ขึ้นมาแสดงสภาพการใช้งานทรัพยากรที่ไม่ยอมให้มีการใช้งานพร้อมกันของ 2 โปรเซส โดยทุกครั้งที่จะมีการเข้าใช้ทรัพยากรนั้น จะต้องมีการตรวจสอบค่าตัวบ่งชี้นี้ว่ามีโปรเซสใดเข้าใช้งานทรัพยากรอยู่หรือไม่ ถ้าไม่มีจึงเข้าใช้งานทรัพยากรนั้น โดยก่อนเข้าทำงาน ต้องเปลี่ยนค่าตัวบ่งชี้เพื่อแสดงสภาพว่ามีโปรเซสกำลังใช้งานทรัพยากรอยู่ ตัวอย่างโปรแกรม คือ การขอเข้าใช้อุปกรณ์เพื่อเขียนข้อมูล เพื่อป้องกันไม่ให้มีการเขียนข้อมูลลงในเรจิสเตอร์ของอุปกรณ์ซ้อนกัน จึงกำหนดตัวบ่งชี้ 2 ตัว คือ dev_in_use เพื่อแสดงว่าอุปกรณ์ได้ถูกเรียกใช้โดยโปรเซสอื่นแล้ว และ dev_busy เพื่อแสดงว่าอุปกรณ์กำลังทำงาน ดังส่วนของโปรแกรมต่อไปนี้

ส่วนของโปรแกรมควบคุมอุปกรณ์

```
while(dev_in_use) /* wait till device is available*)
sleep(&dev_in_use,PSLEP);
dev_in_use = 1; /* allocate the device */
/* initiate the I/O */
dev_busy =1;
/*...*/
oldpri = spl6();
while(dev_busy) /* wait for the transfer to finish */
    sleep(&dev_busy,PSLEP);
splx(oldpri);
/* check for errors */
dev_in_use = 0; /* mark device as available*/
wakeup(&dev_in_use);
/* finish and return */
```

ส่วนของโปรแกรมตอบรับการขัดจังหวะ

```
/* complete the I/O */
dev_busy  =    0;   /* the device is not busy any more */
wakeup(&dev_busy);
```

โปรแกรมดังกล่าวข้างต้น ในส่วนของการตรวจสอบตัวแปร dev_busy จะมีการ

เปลี่ยนลำดับความสำคัญของหน่วยประมวลผล เพื่อป้องกันการขัดจังหวะของอุปกรณ์ก่อนการตรวจสอบตัวบ่งชี้ dev_busy เนื่องจากตัวบ่งชี้นี้อาจถูกเปลี่ยนแปลงค่าได้จากโปรแกรมตอบรับการขัดจังหวะ

5. การตั้งระยะเวลาและการหมดเวลา (timing and timeout routines)

เคอร์เนลได้เตรียมกลไกหลายอย่างเพื่อให้โปรแกรมควบคุมอุปกรณ์สามารถที่จะใช้ในการตรวจวัดเวลาที่ผ่านไป หรือหยุดการทำงานตามระยะเวลาที่กำหนด หรือได้รับการแจ้งถ้าไม่มีสัญญาณขัดจังหวะจากอุปกรณ์ในระยะเวลาที่กำหนด (ซึ่งอาจจะเกิดในกรณีที่อุปกรณ์เสียหาย) กลไกเหล่านี้ได้แก่

5.1 การใช้ฟังก์ชั่น sleep() กับช่องทางสัญญาณ lbolt

ในกรณีที่โปรแกรมควบคุมอุปกรณ์รอคอยเหตุการณ์บางอย่างให้เกิดขึ้นหรือรอให้ทรัพยากรบางอย่างว่างเพื่อใช้งาน และจะไม่มีการส่งสัญญาณขัดจังหวะ โปรแกรมควบคุมอุปกรณ์สามารถจะใช้ฟังก์ชั่น sleep() ได้เช่นกัน โดยใช้ช่องทางสัญญาณ คือเลขที่อยู่ของตัวแปร lbolt โดยเคอร์เนลจะคอยดูแลตัวแปร lbolt โดยจะเพิ่มค่าให้แก่ตัวแปรวินาทีละ 60 ตั้งแต่เวลาที่ระบบปฏิบัติการยูนิกซ์เริ่มทำงาน (bootstrap) เมื่อครบทุก ๆ วินาที เคอร์เนลจะเรียกใช้ฟังก์ชั่น wakeup() โดยใช้ช่องทางสัญญาณเป็นเลขที่อยู่ของตัวแปร lbolt เช่นกัน ดังนั้นโปรแกรมควบคุมอุปกรณ์สามารถจะหยุดพักและถูกปลุกขึ้นมาตรวจสอบเหตุการณ์ทุก ๆวินาที ถ้าเหตุการณ์ยังไม่เกิดขึ้น โปรแกรมควบคุมอุปกรณ์จะวน loop เพื่อเรียกฟังก์ชั่น sleep() ต่อไป

5.2 การป้องกันโปรแกรมควบคุมอุปกรณ์จากการพักการทำงานตลอดไป

คือ เมื่อโปรแกรมควบคุมอุปกรณ์ใช้ฟังก์ชั่น sleep() และรอคอยสัญญาณขัดจังหวะจากอุปกรณ์เพื่อให้โปรแกรมตอบรับสัญญาณขัดจังหวะปลุกโปรแกรมควบคุมอุปกรณ์ขึ้นมาทำงานต่อนั้น มีโอกาสเป็นไปได้ว่า อุปกรณ์อาจจะเสียหายไม่สามารถส่งสัญญาณขัดจังหวะเข้ามาได้ ซึ่งจะทำให้โปรแกรมควบคุมอุปกรณ์พักการทำงานตลอดไป

ยูนิกซ์ได้เตรียมเคอร์เนลฟังก์ชั่นไว้ให้ในกรณีนี้คือฟังก์ชั่น timeout() โดยการเรียกใช้ฟังก์ชั่นต้องส่งพารามิเตอร์ คือ จำนวนเวลา (นับในหน่วยของจังหวะสัญญาณของนาฬิกา) โปรแกรมย่อยที่เคอร์เนลจะต้องเรียกเมื่อเวลาผ่านไปครบตามจำนวนเวลา และข้อมูลที่จะส่งผ่านเป็นพารามิเตอร์ของโปรแกรมย่อย หลังจากที่เวลาผ่านไปครบตามที่ระบุไว้ในฟังก์ชั่น timeout() เคอร์เนลจะเรียกโปรแกรมย่อยที่ระบุในฟังก์ชั่นขึ้นมาทำงาน โปรแกรมย่อยนั้นจะต้องเปลี่ยนค่าบ่งชี้ตัวหนึ่งเพื่อแสดงว่า โปรแกรมควบคุมอุปกรณ์ถูกปลุกจากโปรแกรม timeout() และทำการปลุก (wakeup) โปรแกรมควบคุมอุปกรณ์ที่หยุดพักขึ้นมาเอง โดยใช้ช่องทางสัญญาณเดียวกับที่โปรแกรมควบคุมอุปกรณ์เรียกใช้ในขณะใช้ฟังก์ชั่น sleep()

ในกรณีที่อุปกรณ์ส่งสัญญาณขัดจังหวะเข้ามาได้ ก่อนจะครบกำหนดเวลาที่ตั้งไว้

โปรแกรมย่อยควบคุมอุปกรณ์ที่ทำหน้าที่ตอบรับสัญญาณขัดจังหวะจะต้องเรียกฟังก์ชั่น cantimeout() โดยส่งพารามิเตอร์ เป็นค่าที่ส่งคืนมาจากการเรียกใช้ฟังก์ชั่น timeout() จะเป็นการยกเลิกการตั้งเวลาในฟังก์ชั่น timeout() ที่สั่งไว้

โปรแกรมควบคุมอุปกรณ์สามารถจะใช้ฟังก์ชั่น timeout() หลายครั้ง เนื่องจากการใช้งานฟังก์ชั่น timeout() จะไม่ขึ้นต่อกัน

3.2 การพัฒนาโปรแกรมควบคุมอุปกรณ์ซีดีรอม

โปรแกรมควบคุมอุปกรณ์ซีดีรอมบนระบบปฏิบัติการยูนิกซ์ ได้พัฒนาขึ้นประกอบด้วยโปรแกรมที่เป็นจุดเข้า (entry point) 6 โปรแกรมด้วยกัน แต่ละโปรแกรมมีรายละเอียดการเรียกใช้งานรวมทั้งรหัสลำลอง (Pseudo code) ดังต่อไปนี้

3.1 Srominit()

การเรียกใช้งาน

โปรแกรมย่อยนี้จะถูกเรียกใช้งานโดยเคอร์เนลเพียงครั้งเดียว ในกระบวนการเริ่มต้นการทำงานของระบบปฏิบัติการยูนิกซ์

สภาวะการทำงาน

โปรแกรมย่อยนี้ไม่ได้ถูกเรียกใช้งานในภาวะการทำงานของโปรเซสผู้ใช้ (User process context) ดังนั้นจึงไม่สามารถอ้างถึงข้อมูลในยูเอเรียได้

โปรแกรมย่อยนี้จะต้องไม่เรียกใช้ฟังก์ชั่นระบบต่อไปนี้

- sleep(),delay(), geteblk(), iowait(), longjmp(), physck() , sptfree() เนื่องจากขณะนั้นระบบการติดต่อกับอุปกรณ์ต่าง ๆ ยังติดตั้งไม่สมบูรณ์ ดังนั้นจะไม่มีสัญญาณขัดจังหวะจากอุปกรณ์เข้ามายังโปรแกรมควบคุมอุปกรณ์ ซึ่งจะทำให้ไม่มีเหตุการณ์ (event) ที่จะปลุก (wakeup) โปรแกรมควบคุมอุปกรณ์นี้ให้กลับมาทำงานต่อได้

- timeout() ระยะเวลาที่ใช้จะไม่แน่นอน โดยจะเริ่มต้นนับเวลาเมื่อเคอร์เนลเริ่มต้นการทำงานของระบบนาฬิกาของยูนิกซ์

สำหรับอุปกรณ์แบบสกาสชี่บนระบบปฏิบัติการยูนิกซ์ โปรแกรมย่อยนี้จะยังไม่ส่งคำสั่งเพื่อติดต่อกับอุปกรณ์ (เช่น คำสั่งตรวจสอบสภาวะการทำงานของอุปกรณ์ เป็นต้น) เนื่องจากการส่งคำสั่งติดต่อกับอุปกรณ์สกาสชี่ต้องทำผ่านโปรแกรมควบคุมสกาสชี่โฮสอแดปเตอร์ และอาศัยสัญญาณขัดจังหวะจากอุปกรณ์ เพื่อทราบผลของคำสั่ง ดังนั้นด้วยเหตุผลเดียวกับการไม่ใช้ฟังก์ชั่น sleep() จึงยังไม่ติดต่อกับอุปกรณ์ในโปรแกรมย่อยนี้

รายการข้อมูลที่รับ (argument list)

ไม่มี

การทำงานของโปรแกรม

```
Srominit()
{
    สร้างรายการเชื่อมโยงกองซ้อนของโครงสร้างข้อมูลแบบร้องขอ
    สร้างบัฟเฟอร์ขนาด  2  เคไบต์สำหรับเป็นบัฟเฟอร์ในการถ่ายเทข้อมูล
        ระหว่าง ซีดีรอม และบัฟเฟอร์ของระบบยูนิกซ์
}
```

ค่าข้อมูลส่งกลับ

ไม่มี

3.2 Sromopen(dev, mode, flag)

```
dev_t dev;
int   mode,flag;
```

การเรียกใช้งาน

โปรแกรมย่อยนี้ถูกเรียกใช้งานเมื่อโปรเซสของผู้ใช้ เรียกใช้ฟังก์ชั่นระบบ open() กับอุปกรณ์นี้

สภาวะการทำงาน

โปรแกรมย่อยนี้ถูกเรียกใช้งานในภาวะการทำงานของโปรเซสผู้ใช้ (User process context)

รายการข้อมูลที่รับ

dev เป็นหมายเลขหลักและหมายเลขรองของอุปกรณ์ที่โปรเซสผู้ใช้ขอเริ่มต้นการติดต่อกับอุปกรณ์

mode คือลักษณะการทำงานกับแฟ้มอุปกรณ์ที่ขอเปิด เช่น เปิดเพื่อเขียนข้อมูล เปิดเพื่ออ่านข้อมูล เป็นต้น

flag คือลักษณะการติดต่อกับแฟ้มอุปกรณ์ที่ขอเปิด เช่น เปิดเพื่อติดต่อเป็นแบบบล็อค เปิดเพื่อติดต่อเป็นแบบคาแรคเตอร์ เป็นต้น

รายการข้อมูลรับอื่น ๆ

โปรแกรมย่อยจะพิจารณาค่าตัวแปรแบบสถิต (cdbopened) ด้วยว่ามีโปรเซสอื่นเปิดอุปกรณ์นี้ขึ้นมาใช้หรือไม่ ถ้าเป็นการเปิดอุปกรณ์ขึ้นมาใช้งานครั้งแรกจะแสดงข้อมูลเกี่ยวกับอุปกรณ์ที่เปิดบนจอภาพ

การทำงานของโปรแกรม

```
Sromopen(dev, mode, flag)
dev_t dev;
int mode, flag;
{
```

แยกหมายเลขรองของอุปกรณ์จาก dev;

ค้นหาข้อมูลสถาสซี่โฮสอแดปเตอร์หน่วยที่เกี่ยวข้องกับอุปกรณ์นี้ในตารางโครงแบบ

ค้นหาข้อมูลอุปกรณ์หน่วยนี้ในตารางโครงแบบ

ถ้าไม่พบกำหนดค่าความผิดพลาดเป็น ENODEV ส่งการทำงานกลับคืน

ตรวจสอบความถูกต้องของ mode

ถ้าไม่ถูกต้องกำหนดค่าความผิดพลาดเป็น EACCES ส่งการทำงานกลับคืน

ตรวจสอบความถูกต้องของ flag

ถ้าไม่ใช่เป็นอุปกรณ์แบบบล็อค กำหนดค่าความผิดพลาดเป็น ENOTBLK

ถ้าเป็นอุปกรณ์แบบบล็อค เพิ่มค่าตัวแปร cdbopened เพื่อแสดงว่ามีการเปิดอุปกรณ์นี้ขึ้นมาใช้งาน

ตรวจสอบว่าได้มีการส่งคำสั่งไปยังโฮสอแดปเตอร์เพื่อขอเริ่มต้นการทำงานหรือไม่

ถ้ายังไม่มีการส่งคำสั่งไป

ขอโครงสร้างข้อมูลร้องขอที่ว่าง จากรายการเชื่อมโยงแบบกองซ้อนของโครงสร้างข้อมูลร้องขอ

ส่งคำสั่งไปยังโฮสอแดปเตอร์เพื่อขอเริ่มต้นการทำงาน พร้อมทั้งส่งคำสั่งเพื่อขอข้อมูลความจุสูงสุดของอุปกรณ์ และขนาดบล็อคข้อมูลของอุปกรณ์

ถ้าสถาสซี่โฮสอแดปเตอร์ไม่รับคำสั่ง ให้ทดลองส่งใหม่จนครบ 4 ครั้ง ถ้ายังไม่รับคำสั่งให้กำนดค่าความผิดพลาดเป็น EIO ส่งการทำงานกลับคืน

ถ้าอุปกรณ์เป้าหมายไม่สามารถรับคำสั่ง เนื่องจากเกิดสภาวะแอดเทนชั่น ให้ส่งคำสั่งเริ่มต้นการทำงานโฮสอแดปเตอร์และส่งคำสั่งเดิมซ้ำ

คืนโครงสร้างข้อมูลร้องขอ กลับไปยังรายการเชื่อมโยงแบบกองซ้อน

```
}
```

ค่าข้อมูลส่งกลับ

ไม่มี

3.3 Sromclose(dev, mode, flag)

```
dev_t dev;
int   mode,flag;
```

การเรียกใช้งาน

โปรแกรมย่อยนี้จะถูกเรียกใช้งาน เมื่อโปรเซสสุดท้ายที่เปิดการติดต่อกับอุปกรณ์นี้ขอปิดการติดต่อกับอุปกรณ์

สภาวะการทำงาน

โปรแกรมย่อยนี้ถูกเรียกใช้งานในภาวะการทำงานของโปรเซสผู้ใช้ (User process context)

รายการข้อมูลที่รับ

เหมือนกับตอนเปิดการติดต่อกับอุปกรณ์

การทำงานของโปรแกรม

```
Sromclose()
{
    กำหนดค่าตัวแปรสถิต (cdbopened) ให้เป็น 0 เพื่อแสดงว่าไม่มีโปรเซส
        ใดกำลังใช้งานอุปกรณ์นี้อยู่
}
```

ค่าข้อมูลส่งกลับ

ไม่มี

3.4 Sromstrategy(bp)

```
struct buf * bp
```

การเรียกใช้งาน

โปรแกรมย่อยนี้ถูกเรียกใช้งานโดยเคอร์เนล เพื่อให้ทำการอ่านข้อมูลจากอุปกรณ์แบบบล็อคในตำแหน่งที่ระบุไว้ในบัฟเฟอร์ เข้ามาเก็บยังบัฟเฟอร์ และส่งต่อไปยังโปรเซสผู้ใช้โดยเคอร์เนลอีกทีหนึ่ง

สภาวะการทำงาน

โปรแกรมย่อยนี้ถูกเรียกโดยเคอร์เนล ดังนั้นอาจไม่ได้ถูกเรียกในภาวะการทำงานของโปรเซสผู้ใช้

รายการข้อมูลที่รับ

bp คือ ตัวบ่งชี้ไปยังโครงสร้างข้อมูลของบัฟเฟอร์ที่บรรจุคำสั่งและตำแหน่งของข้อมูลที่โปรเซสผู้ใช้ต้องการ

รายการข้อมูลรับอื่น ๆ

โปรแกรมย่อยนี้ต้องตรวจสอบค่าข้อมูลของตัวแปรในโครงสร้างข้อมูลสถิตว่าอุปกรณ์กำลังทำงานอยู่หรือไม่ ถ้าอุปกรณ์ไม่ได้ทำงานอยู่จะเรียกโปรแกรมย่อย Sromstart() ซึ่งทำหน้าที่สั่งงานให้อุปกรณ์อ่านข้อมูล

การทำงานของโปรแกรม

```
Sramstategy()
{
    ตรวจสอบความถูกต้องของบล็อคข้อมูลที่โปรเซสผู้ใช้ต้องการ
        ถ้าบล็อคข้อมูลไม่อยู่ในช่วงที่ถูกต้องกำหนดค่าความผิดพลาดในบัฟเฟอร์
            คืนบัฟเฟอร์กลับสู่รายการเชื่อมโยงบัฟเฟอร์ของระบบ
            คืนการทำงานไปสู่ระบบ
    ตรวจสอบว่ามีโปรเซสอื่นใช้บัฟเฟอร์กลางอยู่หรือไม่
        ถ้ามีโปรเซสอื่นใช้ให้หยุดการทำงานรอให้จนกว่าบัฟเฟอร์ว่าง
        ถ้าไม่มีโปรเซสอื่นใช้บัฟเฟอร์ เปลี่ยนค่าตัวแปรเพื่อระบุว่าบัฟเฟอร์ถูก
            ใช้แล้ว
    ตรวจสอบว่าบล็อคข้อมูลที่ขออ่านมาเป็นบล็อคข้อมูลเดียวกันกับที่มีการอ่านขึ้น
    มาในครั้งที่แล้วหรือไม่
        ถ้าใช่ทำการถ่ายเทข้อมูลจากบัฟเฟอร์กลางของโปรแกรมควบคุมอุปกรณ์
            (ขนาด 2เคไบต์) ไปยังบัฟเฟอร์ของระบบที่รอข้อมูล(ขนาด 1เคไบต์)
            โดยเลือกส่วนของข้อมูลให้ถูกต้อง
    เปลี่ยนค่าตัวแปรเพื่อระบุว่าไม่มีโปรเซสใดใช้บัฟเฟอร์
    ปลุกโปรเซสอื่นที่รอการใช้บัฟเฟอร์ขึ้นมาทำงานต่อ
    กำหนดสภาวะการทำงานของบัฟเฟอร์นั้นโดยล้างตัวบ่งชี้ที่ระบุว่าบัฟเฟอร์ทำ
        งานเสร็จ
    เพิ่มระดับความสำคัญของหน่วยประมวลผลกลางเพื่อป้องกันการขัดจังหวะของอุปกรณ์
    เรียงลำดับบัฟเฟอร์ใหม่นี้เข้าไปในรายการเชื่อมโยงบัฟเฟอร์ที่รอการทำงาน
    ตรวจสอบว่าอุปกรณ์กำลังทำงานอยู่หรือไม่
```

ถ้าไม่ได้ทำงานเรียกโปรแกรม Sromstart เพื่อทำงานกับบัฟเฟอร์ที่อยู่ในรายการเชื่อมโยง

ลดระดับความสำคัญของหน่วยประมวลผลกลางให้อยู่ระดับเดิม

}

3.5 Sromstart()

การเรียกใช้งาน

โปรแกรมย่อยนี้จะถูกเรียกใช้งานจากทั้งจากโปรแกรมย่อย Sromstrategy และจากโปรแกรมย่อย Sromintr (โปรแกรมตอบรับสัญญาณขัดจังหวะ)

สภาวะการทำงาน

โปรแกรมย่อยนี้อาจไม่ได้ถูกเรียกใช้ในการทำงานของโปรเซสผู้ใช้

รายการข้อมูลรับ

ไม่มี

การทำงานของโปรแกรม

Sromstart()

{

ตรวจสอบว่ามีบัฟเฟอร์ที่รอการทำงานอยู่ในรายการเชื่อมโยงบัฟเฟอร์ที่รอการทำงานหรือไม่

ถ้าไม่มีกำหนดตัวแปรในโครงสร้างข้อมูลสถิตเพื่อระบุว่าอุปกรณ์ไม่ได้ทำงาน พร้อมกับส่งคืนการทำงานกลับ

กำหนดตัวแปรในโครงสร้างข้อมูลสถิตเพื่อระบุว่าอุปกรณ์กำลังทำงาน

ตรวจสอบว่ามีโปรเซสใดใช้บัฟเฟอร์กลางของโปรแกรมควบคุมอุปกรณ์หรือไม่

ถ้ามีโปรเซสอื่นใช้ให้หยุดการทำงานรอให้จนกว่าบัฟเฟอร์ว่าง

ถ้าไม่มีโปรเซสอื่นใช้บัฟเฟอร์ เปลี่ยนค่าตัวแปรเพื่อระบุว่าบัฟเฟอร์ถูกใช้แล้ว

เตรียมบัฟเฟอร์กลางของโปรแกรมควบคุมอุปกรณ์โดยคัดลอกข้อมูลของบัฟเฟอร์ของระบบมาทุกฟิลด์ทั้งหมด

ขอโครงสร้างของข้อมูลร้องขอจากรายการเชื่อมโยงกองซ้อน

เตรียมโครงสร้างข้อมูลร้องขอ

ส่งคำสั่งอ่านข้อมูลไปยังอุปกรณ์เป้าหมาย

ถ้าอุปกรณ์เป้าหมายไม่สามารถรับคำสั่ง เนื่องจากเกิดสภาวะแอดเทนชั่น ให้ส่งคำสั่งซ้ำ

ถ้าสกาสซี่โฮสอแดปเตอร์ไม่รับคำสั่ง ให้ทดลองส่งใหม่จำนวน 4 ครั้ง

ถ้าสกาสซี่โฮสอแดปเตอร์ยังไม่รับคำสั่งแสดงว่ามีความผิดพลาด

ในสกาสซี่โฮสอแดปเตอร์ ให้กำหนดความผิดพลาดเป็น EIO ส่งการทำงานกลับคืน

คืนโครงสร้างข้อมูลร้องขอเข้าสู่รายการเชื่อมโยงกองซ้อนของโครงสร้างข้อมูลแบบร้องขอ

}

3.6 Sromintr(ptr_req)

การเรียกใช้งาน

โปรแกรมย่อยนี้ถูกเรียกใช้งานจากโปรแกรมควบคุมสกาสซี่โฮสอแดปเตอร์ เมื่อได้รับสัญญาณขัดจังหวะจากอุปกรณ์ที่ควบคุมโดยโปรแกรมควบคุมอุปกรณ์นี้

สภาวะการทำงาน

โปรแกรมย่อยถูกเรียกจากภาวะการทำงานแบบขัดจังหวะ

รายการข้อมูลรับ

ตัวแปรบ่งชี้ไปยังโครงสร้างข้อมูลร้องขอ ชุดที่ถูกส่งไปยังโปรแกรมควบคุมสกาสซี่โฮสอแดปเตอร์ เพื่อให้อุปกรณ์ทำงานและส่งสัญญาณขัดจังหวะเข้ามาในครั้งนี้

การทำงานของโปรแกรม

Sromintr()

{

ถ้ามีความผิดพลาดที่สกาสซี่โฮสอแดปเตอร์ รายงานประเภทของความผิดพลาดบนจอภาพ

ถ้ามีความผิดพลาดที่อุปกรณ์เป้าหมาย รายงานประเภทของความผิดพลาดบนจอภาพ ถ้ามีเซนซ์คาต้ารายงานบนจอภาพด้วย

ถ้าไม่มีความผิดพลาดเกิดขึ้น พิจารณาคำสั่งที่ส่ง

ถ้าเป็นคำสั่งในการอ่านข้อมูล โอนถ่ายข้อมูลจากบัฟเฟอร์กลางของโปรแกรมควบคุมอุปกรณ์ไปยังบัฟเฟอร์ของระบบ

จดจำหมายเลขบล็อคข้อมูลที่อ่านในครั้งนี้ในตัวแปรสถิต last_blck

ส่งคืนข้อมูลแก่โปรเซสผู้ใช้

ถ้าเป็นคำสั่งในการตรวจสอบความจุของอุปกรณ์ ให้กำหนดค่าตัวแปรสถิต (max_blck) เป็นค่าเลขหมายบล็อคสูงสุดที่มีอยู่ในอุปกรณ์ที่ได้จากการใช้คำสั่งอ่านความจุ

เปลี่ยนค่าข้อมูลในฟิลด์ของโครงสร้างข้อมูลร้องขอเพื่อแสดงว่าโครงสร้างข้อมูลร้องขอไม่ได้ใช้งาน

ปลุกโปรเซสที่ส่งคำสั่งนี้และใช้ฟังก์ชั่น sleep เพื่อรอการทำงานของอุปกรณ์ตามโครงสร้างข้อมูลร้องขอขึ้นมาทำงานต่อ

เปลี่ยนค่าข้อมูลของตัวแปรเพื่อแสดงว่าไม่มีโปรเซสใดใช้งานบัฟเฟอร์กลางของโปรแกรมควบคุมอุปกรณ์อยู่

ปลุกโปรเซสที่รอใช้บัฟเฟอร์กลางของโปรแกรมควบคุมอุปกรณ์ขึ้นมาทำงาน

ถ้ามีความผิดพลาด พิจารณาคำสั่งที่ส่ง

ถ้าเป็นคำสั่งในการอ่านข้อมูล ให้กำหนดความผิดพลาดเป็น EIO ส่งการทำงานกลับคืน

ถ้าเป็นคำสั่งในการตรวจสอบความจุของอุปกรณ์ กำหนดค่าข้อมูลตัวแปร max_blck เป็น 0

เปลี่ยนค่าข้อมูลของตัวแปรเพื่อแสดงว่าไม่มีโปรเซสใดใช้งานบัฟเฟอร์กลางของโปรแกรมควบคุมอุปกรณ์อยู่

ปลุกโปรเซสที่รอใช้บัฟเฟอร์กลางของโปรแกรมควบคุมอุปกรณ์ขึ้นมาทำงาน

เรียกโปรแกรมย่อย Sromstart เพื่อทำงานต่อ

}