การศึกษาสมรรถนะของลินุกซ์บนระบบฝังตัวสำหรับหน่วยควบคุมการสื่อสารของยูเอวี

นายนครินทร์ เหล่าวัฒนาถาวร

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2549

PERFORMANCE STUDY OF LINUX ON EMBEDDED SYSTEM FOR UAV COMMUNICATION CONTROLLER

Mr. Nakarin Laowattanataworn

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering Program in Electrical Engineering

Department of Electrical Engineering

Faculty of Engineering Chulalongkorn University

Academic Year 2006

หัวข้อวิทยานิพนธ์	การศึกษาสมรรถนะของลินุกซ์บนระบบฝังตัวสำหรับหน่วยควบคุมการสื่อสารของยูเอวี
โดย	นายนครินทร์ เหล่าวัฒนาถาวร
สาขาวิชา	วิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษา	อาจารย์สุวิทย์ นาคพีระยุทธ

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.ดิเรก ลาวัณย์ศิริ)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร.สมชาย จิตะพันธ์กุล)

..... อาจารย์ที่ปรึกษา
(อาจารย์สุวิทย์ นาคพีระยุทธ)

..... กรรมการ
(รองศาสตราจารย์ ดร.ลัญฉกร วุฒิสิทธิกุลกิจ)

..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.มานพ วงศ์สายสุวรรณ)

นครินทร์ เหล่าวัฒนาถาวร : การศึกษาสมรรถนะของลินุกซ์บนระบบฝังตัวสำหรับหน่วยควบคุมการสื่อสารของยูเอวี. (PERFORMANCE STUDY OF LINUX ON EMBEDDED SYSTEM FOR UAV COMMUNICATION CONTROLLER) อ.ที่ปรึกษา : อ.สุวิทย์ นาคพีระยุทธ, จำนวนหน้า 122 หน้า.

วิทยานิพนธ์ฉบับนี้เป็นการศึกษาสมรรถนะของระบบปฏิบัติการลินุกซ์บนระบบฝังตัวที่นำมาทำเป็นตัวควบคุมการสื่อสารของเครื่องบินแบบไม่มีคนบังคับอยู่บนเครื่อง (ยูเอวี) โดยตัวควบคุมการสื่อสารนี้จะต้องรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรม 8 ช่องพร้อมกันที่อัตราบอด 9600 บิตต่อวินาทีและมีปริมาณงาน 38400 บิตต่อวินาที โดยโปรแกรมควบคุมการทำงานของแต่ละช่องทางข้อมูลอนุกรมจะเป็นแบบสายโยงใยซึ่งเป็นอิสระจากกัน ผลการทดสอบบนระบบฝังตัว 2 ชนิดคือพีซี/104 และ RISC คอมพิวเตอร์ ซึ่งมีบัสแบบไอเอสเอที่ใช้สัญญาณขอขัดจังหวะแบบขอบและบัสแบบพีซีไอที่ใช้สัญญาณขอขัดจังหวะแบบระดับตามลำดับ พบว่าพีซี/104 ที่เลือกใช้ไม่สามารถทดสอบใช้งานช่องทางข้อมูลอนุกรมส่วนต่อขยายที่เกิน 4 ช่องทางมาตรฐานได้ เนื่องจากขาดโปรแกรมขับที่เหมาะสมและเมื่อทดสอบเพียง 4 ช่องทางในสภาวะภาระงานปกติที่ 9600 บิตต่อวินาทีด้วยกรรมวิธีรับส่งสัญญาณจะมี excess time สูงสุดถึง 50.4 และ 26.1 มิลลิวินาทีที่การแบ่งเวลา 10 และ 100 มิลลิวินาทีตามลำดับ ซึ่งค่อนข้างมากเมื่อเทียบกับกรอบความที่ใช้รับส่งขนาด 73 มิลลิวินาที ในขณะที่ RISC คอมพิวเตอร์จะมี excess time สูงสุดเพียง 7.3 มิลลิวินาทีที่ปริมาณงาน 70440 บิตต่อวินาที ซึ่งมีค่าไม่เกิน 10 % ของความยาวกรอบความ จึงสามารถนำมาใช้เป็นตัวควบคุมการสื่อสารตามต้องการได้อย่างน่าเชื่อถือ

ภาควิชา วิศวกรรมไฟฟ้า ลายมือชื่อนิสิต นครินทร์ เหล่าวัฒนาถาวร
สาขาวิชา วิศวกรรมไฟฟ้า ลายมือชื่ออาจารย์ที่ปรึกษา
ปีการศึกษา 2549

4770314321 : MAJOR ELECTRICAL ENGINEERING

KEYWORD : KERNEL / LINUX / EMBEDDED

NAKARIN LAOWATTANATAWORN : PERFORMANCE STUDY OF LINUX ON EMBEDDED SYSTEM FOR UAV COMMUNICATION CONTROLLER. THESIS ADVISOR : SUVIT NAKPEERAYUTH, 122 pp.

This thesis studied the performance of Linux OS on an embedded system, communication controller for unmanned aerial vehicle (UAV). The communication controller must handle 8 serial ports at 9600 bps concurrently with throughput 38400 bps. The control programs for each port will be multi-thread. The performances of two programming methods were evaluated, polling and signaling. The performances were tested on two embedded platforms, PC/104 with edge interrupt ISA bus and RISC based computer with level interrupt PCI bus. The selected PC/104 can not handle the extended serial ports above 4 standard ports due to lack of proper driver. When testing with 4 ports at 9600 bps under typical load, the signaling method has 50.4 and 26.1 ms maximum excess time for 10 and 100 ms time slice respectively. This is quite large for 73 ms frame size. While the RISC based computer has only 7.3 ms maximum excess time at 70440 bps throughput which is less than 10 percent of the frame size. This can be reliably used as a communication controller under the working condition.

Department Electrical Engineering Student's Signature นครินทร์ เหล่าวัฒนาตาร

Field of study Electrical Engineering Advisor's Signature

Academic Year 2006

กิตติกรรมประกาศ

ขอขอบคุณ อ.สุวิทย์ นาคพีระยุทธ ที่ได้ช่วยเหลือด้านคำแนะนำ จัดหาเอกสาร เครื่องมือ และอุปกรณ์ ตลอดจนถึงการตรวจวิทยานิพนธ์ฉบับนี้

สารบัญ

หน้า

สารบัญตาราง

ตาราง **หน้า**

สารบัญรูป

รูป **หน้า**

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

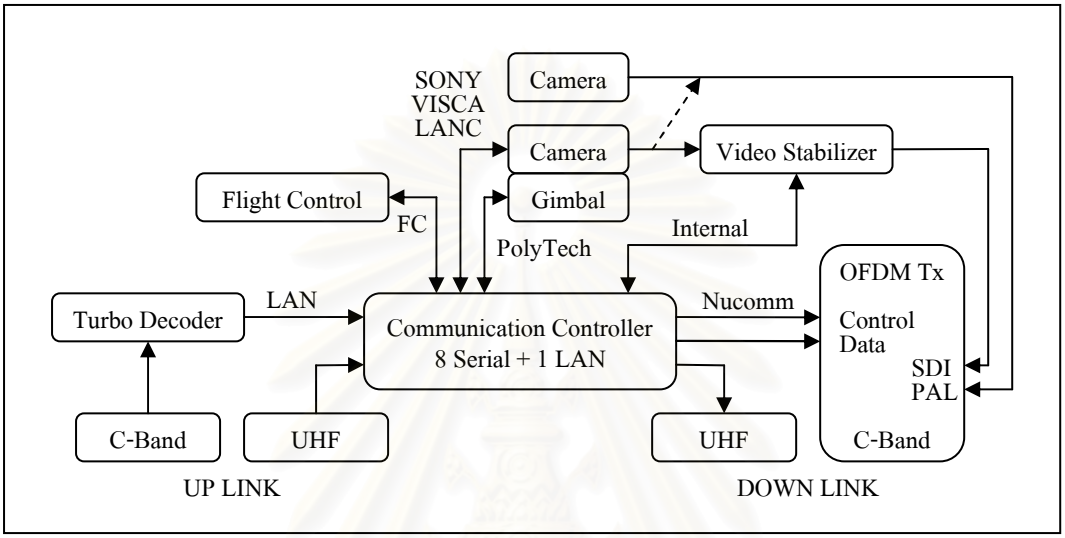
ต้องการพัฒนาและสร้างเครื่องบินแบบไม่มีคนบังคับอยู่บนเครื่อง โดยเครื่องบินลำนี้มีเป้าหมายหรือหน้าที่หลัก ๆ คือบินไปถ่ายภาพและส่งภาพที่ถ่ายนั้นกลับมาให้ภาคพื้นในแบบทันที ทำให้เครื่องบินลำนี้ต้องมีกล้องและอุปกรณ์ควบคุมเครื่องบินต่าง ๆ อยู่หลายส่วนและเนื่องจากคลื่นวิทยุที่ใช้รับส่งข้อมูลระหว่างภาคพื้นกับเครื่องบินมีอยู่จำกัด ข้อมูลที่จะติดต่อสื่อสารระหว่างภาคพื้นกับตัวเครื่องจึงต้องใช้คลื่นวิทยุร่วมกันทำให้เครื่องบินลำนี้ต้องมีอุปกรณ์ที่ทำหน้าที่กระจายข้อมูลที่รับจากภาคพื้นเพื่อส่งไปยังอุปกรณ์ต่าง ๆ และในขณะเดียวกันอุปกรณ์ตัวนี้ก็ต้องทำการรวมข้อมูลจากอุปกรณ์ต่าง ๆ ส่งกลับไปยังภาคพื้นได้อย่างถูกต้องและรวดเร็ว

จากความต้องการเบื้องต้นอุปกรณ์ดังกล่าวควรจะเป็นระบบฝังตัวจึงได้แบ่งการพิจารณาออกเป็น 2 ส่วนคือฮาร์ดแวร์และซอฟต์แวร์ ในส่วนของฮาร์ดแวร์จำเป็นจะต้องมีขนาดเล็ก น้ำหนักเบา กินไฟน้อย มีช่องทางเพียงพอสำหรับการสื่อสาร สามารถตั้งใหม่ (reset) ตัวเองได้เมื่อระบบล้มเหลว ฯลฯ ซึ่งทางผู้ทดสอบได้เลือกทดสอบกับฮาร์ดแวร์ 2 แพลตฟอร์มคือพีซี/104 รุ่น PCM-3341 + PCM3643 ของ Advantech และ RISC รุ่น UC-7408 ของ MOXA และเนื่องจากระบบฝังตัวนี้ต้องเชื่อมต่อกับอุปกรณ์อื่นอีกหลายส่วนทำให้งานที่ทำเป็นแบบหลายภาระงาน ผู้ทดสอบจึงได้เลือกให้ระบบฝังตัวทำงานโดยมีระบบปฏิบัติการเป็นผู้จัดสรรเวลาให้แต่ละงาน จึงเลือกลินุกซ์มาเป็นระบบปฏิบัติการเพราะมีข้อดีหลายประการเช่นลินุกซ์ทำงานได้รวดเร็วและเป็นระบบปฏิบัติการฟรีภายใต้จีพีแอล สามารถปรับแต่งได้ตามสภาพแวดล้อม มีเสถียรภาพ ฯลฯ ซึ่งตัวควบคุมนี้จะต้องรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรมพร้อมกันทั้ง 8 ช่องที่อัตราบอด 9600 บิตต่อวินาทีโดยมีปริมาณงาน 38400 บิตต่อวินาทีและมีระยะเวลาการประมวลผลที่ยอมรับได้

แต่ว่าบุคลากรที่มีความรู้ความเข้าใจในระบบฝังตัวยังมีจำนวนน้อยถึงแม้ว่าศาสตร์ดังกล่าวจะมีการใช้งานมานานแล้วแต่ก็จำกัดอยู่ในวงแคบเพราะในอดีตอุปกรณ์ที่ใช้ทดสอบมีราคาสูงและเครื่องมือในการพัฒนาก็มีน้อยและไม่แพร่หลายทำให้ผู้ที่ต้องการศึกษามีจำนวนน้อยตามไปด้วย แต่ในปัจจุบันอุปกรณ์ต่าง ๆ มีราคาถูกลงมากและเครื่องมือที่ใช้พัฒนาก็มีเพิ่มมากขึ้นโดยหาได้จากอินเทอร์เน็ตและจากข้อดีดังกล่าวทำให้ปัจจุบันศาสตร์ทางด้านนี้ถูกนำไปใช้งานอย่างกว้างขวางส่งผลให้มีความต้องการบุคลากรทางด้านนี้เพิ่มมากขึ้นและเพื่อให้การพัฒนาเป็นไปอย่างรวดเร็วและต่อเนื่องจึงจำเป็นที่จะต้องสร้างบุคลากรรุ่นใหม่ให้มีความรู้ความสามารถมาเป็นกำลังสำคัญในอนาคตสืบไป

1.2 วัตถุประสงค์ของการวิจัย

พัฒนาระบบฝังตัวเพื่อนำไปใช้เป็นตัวควบคุมการสื่อสารของเครื่องบินขนาดเล็กแบบไม่มีคนบังคับอยู่บนเครื่องซึ่งตัวควบคุมการสื่อสารนี้มีการเชื่อมต่อกับอุปกรณ์หลายส่วน แต่หน้าที่โดยรวมคือรับข้อมูลจากภาคพื้นแล้วส่งออกไปตามช่องทางขาออกให้ถูกต้อง และในขณะเดียวกันก็ต้องรับข้อมูลจากอุปกรณ์ต่าง ๆ เพื่อส่งไปยังภาคพื้นโดยมีรายละเอียดดังรูป 1.1 และตาราง 1.1



รูป 1.1 การเชื่อมต่อระหว่างตัวควบคุมการสื่อสารกับส่วนต่างๆ

ตาราง 1.1 การเชื่อมต่อระหว่างตัวควบคุมการสื่อสารกับส่วนต่าง ๆ

กลุ่ม	เชื่อมต่อกับ	หน้าที่	พอร์ต
Up Link	Turbo Decoder	รับข้อมูล C-Band	อีเทอร์เน็ต
Up/Down Link	UHF	รับและส่งข้อมูล UHF	อนุกรม 1
Flight Control	Flight Control	บังคับเครื่องบิน	อนุกรม 2
Pay Load	Camera	บังคับตัวกล้อง เช่น ซูมภาพ	อนุกรม 3
	Gimbal	บังคับขากล้อง เช่น หมุนกล้อง	อนุกรม 4
	Video Stabilizer	ควบคุม Video Stabilizer	อนุกรม 5
	Control ของ OFDM Tx	เลือก Channel Coderate BW	อนุกรม 6
Down Link	Data ของ OFDM Tx	ส่งข้อมูล C-Band	อนุกรม 7
สำรอง	สำรอง	สำรอง	อนุกรม 8

โดยความต้องการเบื้องต้นของตัวควบคุมการสื่อสารที่จะพัฒนานี้ต้องทำงานได้ในสภาวะภาระงานปกติที่อัตราบอด 9600 บิตต่อวินาทีและมีปริมาณงาน 38400 บิตต่อวินาที

1.3 ขอบเขตของการวิจัย

ทดสอบประสิทธิภาพการรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรมแบบทันทีในระบบฝังตัวบนแพลตฟอร์ม 2 แบบที่แตกต่างกันคือแพลตฟอร์มที่ใช้สถาปัตยกรรมแบบ x86 และแพลตฟอร์มที่ใช้สถาปัตยกรรมแบบ ARM โดย x86 จะทดสอบกับพีซี/104 รุ่น PCM-3341 + PCM3643 ของ Advantech ที่ช่องทางข้อมูลอนุกรมอยู่บนบัสแบบไอเอสเอที่ใช้สัญญาณขอขัดจังหวะแบบขอบ (edge interrupt) ส่วน ARM จะทดสอบกับ RISC รุ่น UC-7408 ของ MOXA ที่ช่องทางข้อมูลอนุกรมอยู่บนบัสแบบพีซีไอที่ใช้สัญญาณขอขัดจังหวะแบบระดับ (level interrupt) ซึ่งข้อแตกต่างของสัญญาณขอขัดจังหวะแบบขอบและแบบระดับคือสัญญาณขอขัดจังหวะแบบขอบจะถูกตรวจสอบในจังหวะที่มีการเปลี่ยนแปลงของระดับสัญญาณไฟฟ้าเท่านั้น ถ้าหากผู้รับสัญญาณขอขัดจังหวะตรวจจังหวะการเปลี่ยนแปลงของสัญญาณไฟฟ้าผิดพลาดก็จะทำให้สัญญาณนั้นสูญหายไปหรือในกรณีที่มีอุปกรณ์หลายอุปกรณ์ใช้ไออาร์คิวร่วมกันและแต่ละอุปกรณ์ส่งสัญญาณขอขัดจังหวะพร้อมกันก็มีโอกาสที่สัญญาณขอขัดจังหวะจะซ้อนทับกันพอดีซึ่งการทำงานก็ขึ้นกับโปรแกรมขับว่าจะให้บริการกับอุปกรณ์อย่างไรเช่นให้บริการกับอุปกรณ์ตัวหนึ่งตัวใดแล้วหยุดซึ่งทำให้อุปกรณ์ตัวอื่นไม่ได้รับบริการ หรืออีกแบบคือโปรแกรมขับต้องทราบก่อนว่ามีอุปกรณ์ใดบ้างที่ใช้ไออาร์คิวเส้นนี้และเมื่อมีสัญญาณขอขัดจังหวะเข้ามาโปรแกรมขับก็ต้องให้บริการกับอุปกรณ์ทุกตัวที่ใช้ไออาร์คิวเส้นดังกล่าว ส่วนการขอขัดจังหวะแบบระดับนั้นมีข้อดีเหนือกว่าแบบขอบเพราะผู้รับสัญญาณจะตรวจสอบสัญญาณขอขัดจังหวะที่ระดับของสัญญาณไฟฟ้าที่จะค้างอยู่ตลอดเวลาจนกว่าอุปกรณ์ที่ส่งสัญญาณขอขัดจังหวะจะได้รับการบริการจากโปรแกรมขับ โดยโปรแกรมขับจะวนให้บริการกับอุปกรณ์ที่ใช้ไออาร์คิวดังกล่าวไปจนกว่าระดับของสัญญาณขอขัดจังหวะจะไม่แอ็กทิฟ ซึ่งระบบนี้สามารถรับประกันได้ว่าอุปกรณ์ทุกตัวที่ส่งสัญญาณขอขัดจังหวะไปแล้วจะได้รับการบริการจากโปรแกรมขับแน่นอนแม้ว่าจะใช้ไออาร์คิวร่วมกันกับอุปกรณ์อื่นก็ตาม โดยมีเป้าหมายของการทดสอบดังนี้

1. ที่ภาระงานปกติคือระบบสามารถรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรมพร้อมกันทั้ง 8 ช่องที่อัตราบอด 9600 บิตต่อวินาทีและมีปริมาณงาน 38400 บิตต่อวินาที โดยระยะเวลาการประมวลผลยังคงยอมรับได้

2. ที่ภาระงานสูงสุดคือระบบสามารถรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรมพร้อมกันทั้ง 8 ช่องอยู่ตลอดเวลาที่อัตราบอด 9600 บิตต่อวินาที

3. หาขีดจำกัดของระบบเมื่อรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรมพร้อมกันทั้ง 8 ช่อง

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. หน่วยควบคุมการสื่อสารของยูเอวี
2. ความรู้ในการปรับแต่งลินุกซ์ให้เหมาะสมกับทรัพยากรที่มีและงานที่ทำ
3. แพลตฟอร์มช่วยการพัฒนาสำหรับงานระบบฝังตัวแบบทันทีที่ใกล้เคียงอื่นๆ

1.5 วิธีดำเนินการวิจัย

1. ศึกษาข้อมูลของระบบฝังตัวที่จะใช้เป็นฮาร์ดแวร์ของการทดสอบ
2. ศึกษาลินุกซ์ที่จะนำมาใช้เป็นระบบปฏิบัติการในระบบฝังตัว
3. ปรับเปลี่ยนเคอร์เนลของลินุกซ์ที่จะนำไปเป็นระบบปฏิบัติการบน PCM-3341 + PCM-3643 เพื่อลดขนาดและเพิ่มความเร็วให้ระบบ
4. เขียนโปรแกรมทดสอบสมรรถนะการรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรมบน PCM-3341 + PCM-3643 ที่มีลินุกซ์ที่ผู้ทดสอบประกอบขึ้นเองเป็นระบบปฏิบัติการ
5. นำโปรแกรมในข้อ 4 ไปทดสอบบน UC-7408 ที่มีลินุกซ์ติดตั้งมาแล้วเพื่อทำการเปรียบเทียบ
6. เก็บผลนำมาสรุปและเขียนวิทยานิพนธ์

บทที่ 2

แนวคิดและทฤษฎี

2.1 ระบบฝังตัว (1)

2.1.1 บทนำระบบฝังตัว

ระบบฝังตัวคือคอมพิวเตอร์ที่รวมเอาอุปกรณ์ในความควบคุมทุกชิ้นมาอยู่บนระบบเดียวกันเพื่อวัตถุประสงค์พิเศษบางอย่างซึ่งแตกต่างกับคอมพิวเตอร์อเนกประสงค์อย่างเครื่องคอมพิวเตอร์ส่วนบุคคลเพราะระบบฝังตัวมักสร้างมาทำภารกิจตามที่ได้กำหนดไว้และภารกิจที่ทำก็เป็นงานที่ไม่หนักมาก

เมื่อระบบถูกกำหนดภารกิจล่วงหน้าแล้วทำให้ผู้ออกแบบสามารถออกแบบระบบให้เหมาะสมกับงานได้มากที่สุดเช่นลดขนาดและราคาของระบบเพราะการออกแบบระบบฝังตัวแต่ละครั้งนิยมจะผลิตออกมาเป็นจำนวนมาก โดยตัวอย่างของอุปกรณ์ที่มีระบบฝังตัวเป็นส่วนประกอบได้แก่

- อุปกรณ์เพื่ออำนวยความสะดวกเช่น เครื่องคิดเลข เครื่องพีดีเอ กล้องดิจิทัล โทรศัพท์เคลื่อนที่ โทรทัศน์ เครื่องเล่น/บันทึกวีซีดี/ดีวีดี เครื่องเล่นเอ็มพีต่าง ๆ เครื่องเล่นเกมส์ เครื่องเอทีเอ็ม เตาไมโครเวฟ เครื่องปรับอากาศ ระบบกล้องวงจรปิด เครื่องซักล้าง รวมถึงอุปกรณ์จัดเส้นทางและเครื่องพิมพ์ที่ต่อกับคอมพิวเตอร์
- อุปกรณ์เพื่ออำนวยความปลอดภัยเช่น สัญญาณไฟจราจร ระบบนำทางและหน่วยควบคุมทางการบิน หน่วยควบคุมเครื่องยนต์และหน่วยควบคุมระบบห้ามล้อในรถยนต์

2.1.2 ประวัติระบบฝังตัว

ในปี ค.ศ.1960 คอมพิวเตอร์สามารถคำนวณวิเคราะห์และประมวลผลข้อมูลด้วยความเร็วสูงแต่ทว่า ณ เวลานั้นคอมพิวเตอร์ยังมีราคาที่ถือว่าสูงมากอีกทั้งการโปรแกรมยังมีความซับซ้อนทำให้จำนวนของผู้ที่ต้องการศึกษาเรียนรู้มีจำนวนน้อย ต่อมาเมื่อมีการพัฒนาตัวควบคุมแบบโปรแกรมได้ให้กับคอมพิวเตอร์จึงทำให้การโปรแกรมทำได้ง่ายขึ้นและตัวควบคุมแบบโปรแกรมได้นี้ก็กลายมาเป็นส่วนหนึ่งของระบบฝังตัว

ระบบฝังตัวที่โดดเด่นในช่วงยุค 1960 ได้แก่ระบบนำทางของยานอะพอลโลที่พัฒนาขึ้นโดย Charles Stark Draper จากห้องปฏิบัติการของเอ็มไอที ณ ตอนที่เริ่มโครงการอะพอลโลระบบนำทางถือว่าเป็นส่วนที่มีความเสี่ยงสูงที่สุดเพราะการใช้วงจรรวมเป็นชิ้นเดียวอีกทั้งมีความ

พยายามที่จะลดขนาดและน้ำหนักของวงจรที่ยังถือเป็นเรื่องใหม่ในขณะนั้นยิ่งเป็นการเพิ่มความเสี่ยงให้ระบบสูงขึ้น

การผลิตระบบฝังตัวที่มีจำนวนผลิตสูงเป็นครั้งแรกคือการผลิตระบบนำทาง “Autonetics D-17” สำหรับ “Minuteman missile” ในปี ค.ศ.1961 ซึ่งสร้างจาก “discrete transistor digital circuit logic” และใช้จานบันทึกแบบแข็งเป็นหน่วยความจำหลัก ต่อมาในปี ค.ศ.1966 มีการผลิต “Minuteman II” โดยเปลี่ยนจาก D-17 ไปใช้ชิปแบบใหม่ทำให้ราคาผลิตลดจาก $1000 ต่อชิ้น เหลือเพียง $3 ต่อชิ้น

ไมโครโพรเซสเซอร์ในยุคแรกคืออินเทล 4004 แม้จะสามารถคำนวณและประมวลผลในระบบขนาดเล็กได้แต่ต้องอาศัยหน่วยความจำภายนอกและต้องอาศัยชิปอื่นเพื่อให้ระบบทำงานได้ จนถึงกลางปี ค.ศ.1980 ส่วนประกอบของระบบที่เคยอยู่ภายนอกเกือบทั้งหมดถูกนำมาสร้างรวมอยู่บนชิปเดียวกันกับหน่วยประมวลผลและเรียกวงจรรวมนี้ว่า “เครื่องควบคุมระดับไมโคร” (microcontroller) ซึ่งเป็นที่นิยมใช้งานกันอย่างแพร่หลายในระบบฝังตัว

2.1.3 **ลักษณะของระบบฝังตัว**

ระบบฝังตัวถูกออกแบบมาให้ทำตามภารกิจที่ระบุไว้ ซึ่งบางภารกิจมีเงื่อนไขทางด้านเวลามาเกี่ยวข้องเช่นภารกิจที่ให้ทำต้องเสร็จภายในระยะเวลาที่กำหนดโดยเรียกระบบเช่นนี้ว่า “ทันที” อาจเพื่อเหตุผลทางด้านความปลอดภัยแต่ก็มีภารกิจอื่น ๆ ในระบบฝังตัวที่ไม่ต้องการการตอบสนองต่อคำสั่งทันทีซึ่งระบบแบบหลังนี้ทำให้ลดราคาฮาร์ดแวร์ของระบบได้ระดับหนึ่ง

ซอฟต์แวร์สำหรับระบบฝังตัวมักถูกเรียกว่า “เฟิร์มแวร์” และจะเก็บแบบให้อ่านได้อย่างเดียวภายในชิปหน่วยความจำแฟลชมากกว่าที่จะเก็บในหน่วยขับแผ่นบันทึก โดยซอฟต์แวร์มักจะดำเนินงานภายใต้ทรัพยากรทางด้านฮาร์ดแวร์ที่จำกัดเช่น หน่วยความจำมีขนาดเล็ก ไม่มีแผงแป้นอักขระ ไม่มีจอภาพ แต่ระบบฝังตัวที่อยู่ภายในเครื่องใด ๆ ที่ต้องดำเนินการติดต่อกันเป็นปีมักถูกคาดหวังว่าระบบจะต้องไม่มีข้อผิดพลาดเกิดขึ้นเลยหรือถ้ามีข้อผิดพลาดเกิดขึ้นแล้วระบบฝังตัวก็ต้องสามารถกู้ระบบได้ด้วยตัวเองดังนั้นซอฟต์แวร์สำหรับระบบฝังตัวต้องถูกพัฒนาและทดสอบอย่างละเอียดมากกว่าซอฟต์แวร์ที่ใช้กับเครื่องพีซี ชิ้นส่วนของเครื่องที่เคลื่อนที่ได้เช่นหน่วยขับแผ่นบันทึก สวิตช์ ปุ่มกด มักจะหลีกเลี่ยงไม่นำมาใช้ในระบบฝังตัวส่วนการกู้ระบบอาจทำได้ด้วยตัวจับเวลา “วอตช์ดอก” (watchdog) ที่จะตั้งระบบใหม่ถ้าหากคาบซอฟต์แวร์แจ้งไปยังวอตช์ดอก

2.1.4 **แพลตฟอร์มซีพียู**

สถาปัตยกรรมซีพียูที่ใช้ในระบบฝังตัวมีหลากหลายสถาปัตยกรรมเช่น ARM architecture, MIPS architecture, Coldfire68k, PowerPC, x86 architecture, PIC

microcontrollers, 8051/8751, Atmel AVR, Renesas H8, SuperH(SH), V850, FR-V, M32R, eZ80, Z8 ฯลฯ ซึ่งมากกว่าสถาปัตยกรรมของซีพียูในเครื่องพีซีที่มีแข่งขันอยู่ไม่กี่สถาปัตยกรรม

หน่วยประมวลผลที่ใช้แบ่งได้ 2 ประเภทคือไมโครโพรเซสเซอร์ที่ไม่มีอุปกรณ์รอบข้างติดมาบนแกนซิลิคอนและอีกแบบคือเครื่องควบคุมระดับไมโครที่มีอุปกรณ์รอบข้างบางส่วนติดมาด้วยบนแกนซิลิคอน

พีซี/104 และพีซี/104+ เป็นฮาร์ดแวร์สำหรับระบบฝังตัวที่อาจถือว่าผลิตมาให้ผู้ใช้ได้เลือกใช้งานกับระบบฝังตัวที่มีปริมาณชิ้นงานน้อย ๆ โดยไม่ต้องไปออกแบบวงจรที่ดีที่สุดสำหรับระบบฝังตัวเพราะอาจไม่คุ้มกับการออกแบบภายใต้จำนวนชิ้นงานน้อยชิ้นซึ่งพีซี/104 และพีซี/104+ ก็ได้รับความนิยมในการใช้งานพอสมควรเพราะถือว่ามีขนาดเล็กและมีหลายแบบให้เลือก ร่วมทั้งยังสามารถใช้งานกับดอส ลินุกซ์ เน็ตบีเอสดี ระบบปฏิบัติการแบบทันทีฝังตัวโดยเฉพาะเช่น QNX, Inferno, Wind River's VxWorks or Green Hill's Intergrity

สำหรับระบบฝังตัวที่ต้องผลิตจำนวนมากชิ้นมักจะออกแบบเพื่อหาระบบหรือหาวงจรที่ดีที่สุดทั้งทางด้านขนาด น้ำหนัก ราคา ฯลฯ แล้วจึงผลิตออกมาเป็นระบบบนชิป (system on chip) โดยเรียกว่า “วงจรรวมจำเพาะงานประยุกต์” (ASIC : application-specific integrated circuit)

2.1.5 **อุปกรณ์รอบข้าง**

ระบบฝังตัวมักจะต้องมีการสื่อสารกับอุปกรณ์ภายนอกอื่นเพื่อให้ระบบทำงานได้แต่ที่นิยมใช้ก็เป็นช่องทางปกติที่ใช้อยู่กับเครื่องมือหรือพีซีในยุคนั้น ๆ ตัวอย่างส่วนต่อประสานกับอุปกรณ์อื่นได้แก่ อาร์เอส-232 อาร์เอส-422 อาร์เอส-485 อาร์เจ-45 ยูเอสบี บลูทูท จีพีไอบี ซีเอเอ็น ทีพียู เอสเอสซี/อีเอสเอสไอ ฯลฯ

2.1.6 **เครื่องมือพัฒนาระบบฝังตัว**

การพัฒนาซอฟต์แวร์สำหรับระบบฝังตัวก็มีลักษณะคล้ายการพัฒนาซอฟต์แวร์ปกติทั่วไป คือมีการใช้คอมไพเลอร์ ภาษาแอสเซมบลีและโปรแกรมตรวจแก้จุดบกพร่อง แต่ระบบฝังตัวบางระบบอาจต้องใช้เครื่องมือจำเพาะบางอย่างเข้ามาช่วยพัฒนาซอฟต์แวร์เช่น

- ตัวเลียนแบบภายในวงจรหรือไอซีอี (ICE : in-circuit emulator) คืออุปกรณ์ฮาร์ดแวร์ที่ใช้แทนที่หรือเสียบเข้ากับไมโครโพรเซสเซอร์โดยอุปกรณ์ดังกล่าวจะอำนวยความสะดวกในการโหลดและแก้จุดบกพร่องของรหัสที่ใช้ทดสอบในระบบ

- โปรแกรมตรวจแก้จุดบกพร่องภายในวงจรหรือไอซีดี (ICD : in-circuit debugger) คืออุปกรณ์ฮาร์ดแวร์ที่ใช้แทนที่หรือเสียบเข้ากับไมโครโพรเซสเซอร์ผ่านทางส่วนต่อประสาน JTAG

หรือ NEXUS โดยอุปกรณ์ดังกล่าวจะอำนวยความสะดวกในการโหลดและแก้จุดบกพร่องของรหัสที่ใช้ทดสอบในระบบ

- โปรแกรมอรรถประโยชน์ที่ใช้โปรแกรมผลรวมตรวจสอบหรือซีอาร์ซี

- สำหรับระบบที่มีการใช้ดีเอสพี การพัฒนาหรือจำลองทางคณิตศาสตร์อาจต้องใช้ MathCad หรือ Mathematica

- คอมไพเลอร์และโปรแกรมเชื่อมโยงเฉพาะอาจต้องใช้เพื่อพัฒนาซอฟต์แวร์ให้เหมาะที่สุดสำหรับฮาร์ดแวร์เฉพาะนั้น ๆ

- ระบบฝังตัวอาจมีภาษาหรือเครื่องมือออกแบบพิเศษเป็นของตัวเองหรืออาจเป็นเครื่องมือที่ใช้เสริมกับภาษาที่มีอยู่เพื่อพัฒนาซอฟต์แวร์ให้ใช้งานได้หรือให้ใช้ได้ดีขึ้น
เครื่องมือในการพัฒนาซอฟต์แวร์อาจมาจากหลายแหล่งเช่น

- บริษัทที่ทำซอฟต์แวร์สำหรับระบบฝังตัวโดยเฉพาะ

- จีเอ็นยู (GNU) ซอฟต์แวร์

- ซอฟต์แวร์สำหรับพีซีทั่วไปอาจใช้ในระบบฝังตัวได้หากหน่วยประมวลผลที่ใช้มีความใกล้เคียงกัน

2.1.7 **การแก้จุดบกพร่อง**

การแก้จุดบกพร่องในระบบฝังตัวมีหลายระดับขึ้นกับเครื่องมืออำนวยความสะดวกที่หาได้ โดยมีตั้งแต่ระดับแอสเซมบลีหรือระดับรหัสต้นฉบับที่แก้จุดบกพร่องด้วยตัวเลียนแบบภายในวงจรหรือโปรแกรมตรวจแก้จุดบกพร่องภายในวงจรซึ่งจะส่งเอาต์พุตออกมาทางช่องทางแก้จุดบกพร่องหรือส่วนต่อประสาน JTAG หรือ NEXUS ไปจนถึงระดับที่จำลองสิ่งแวดล้อมให้ดำเนินงานบนพีซี

2.1.8 **ความน่าเชื่อถือ**

ระบบฝังตัวบางระบบเมื่อใช้งานไปแล้วมักจะไม่สามารถปิดระบบเพื่อทำการซ่อมแซมส่วนที่เสียได้หรือบางระบบอาจยากลำบากในการเข้าไปซ่อมแซมเช่นระบบที่ทำงานอยู่ในอวกาศ เคเบิลใต้น้ำหรือระบบที่ต้องดำเนินงานอย่างต่อเนื่องในช่วงเวลาทำงานเพื่อเหตุผลด้านความปลอดภัยทั้งทางชีวิตและทรัพย์สินเช่นระบบนำทางบนเครื่องบิน ระบบควบคุมสารเคมีอันตรายในโรงงาน ระบบควบคุมเครื่องยนต์บนเครื่องบิน ระบบทางการเงิน ระบบทางการแพทย์ ฯลฯ ซึ่งโดยทั่วไประบบที่กล่าวมาจะต้องมีระบบสำรองที่สามารถทำงานแทนที่ระบบหลักได้ทันทีที่ระบบหลักไม่สามารถทำงานต่อไปได้เพราะอาจมีความเสียหายตามมาถ้าระบบฝังตัวไม่สามารถทำงานได้ซึ่งความเสียหายจะมากน้อยแตกต่างกันไปตามงานที่ทำและที่สำคัญไม่ควรใช้ระบบฝังตัวที่ยัง

ไม่ปลอดภัยกับระบบที่เกี่ยวข้องกับความปลอดภัยเพราะหากระบบฝังตัวล้มเหลวแล้วอาจส่งผลกระทบร้ายแรงต่อผู้เกี่ยวข้อง

2.1.9 ระบบควบคุมการขัดจังหวะ

ระบบฝังตัวบางระบบใช้การขัดจังหวะเพื่อแจ้งเหตุการณ์ต่าง ๆ ให้ระบบทราบซึ่งสัญญาณขัดจังหวะถูกสร้างมาจากหลาย ๆ ทางเช่นจากตัวจับเวลาหรือตัวควบคุมช่องทางไอ/โอต่าง ๆ ฯลฯ คุณสมบัติที่ต้องการในการขัดจังหวะคือระยะเวลาการตอบสนองของระบบต่อสัญญาณขัดจังหวะต้องรวดเร็ว การจัดกระทำต่อสัญญาณขัดจังหวะต้องสั้นและง่ายเพื่อลดเวลาแฝงการขัดจังหวะให้น้อยที่สุด แต่สัญญาณขัดจังหวะอาจต้องต่อคิวเพื่อรอการประมวลผลที่เหมาะสม

2.2 ลินุกซ์ (2)

2.2.1 บทนำลินุกซ์

ลินุกซ์ถูกถอดแบบมาจากระบบปฏิบัติการยูนิกซ์โดยการเขียนของคุณ Linus Torvalds ตั้งแต่ปี ค.ศ. 1991 โดยได้รับการช่วยเหลือจากแฮ็กเกอร์ผ่านระบบเครือข่าย โดยมีเป้าหมายให้สอดคล้องกับข้อกำหนดของโพสิกซ์และยูนิกซ์ดังนั้นลินุกซ์จึงมีคุณลักษณะเด่น ๆ ทุกอย่างที่ยูนิกซ์ฉบับเต็มพึงมีตัวอย่างเช่น ระบบหลายภาระกิจแบบแท้จริง หน่วยความจำเสมือน การใช้คลังโปรแกรมร่วมกัน คำขอการโหลด การใช้ copy-on-write ที่กระทำการได้ การจัดการหน่วยความจำที่เหมาะสม เครือข่ายหลายกองซ้อนทั้งไอพีวี4 (IPv4) และไอพีวี6 (IPv6) โดยการแพร่กระจายของลินุกซ์นั้นจะอยู่ภายใต้จีเอ็นยูจีพีแอล (GNU General Public License)

2.2.2 ฮาร์ดแวร์ที่ลินุกซ์ดำเนินงานได้

แม้ว่าเริ่มแรกลินุกซ์จะถูกพัฒนาสำหรับเครื่อง 32 บิต x86 พีซี (386 หรือสูงกว่า) แต่ทุกวันนี้ลินุกซ์สามารถดำเนินงานได้บนหลากหลายสถาปัตยกรรมเช่น the Compaq Alpha AXP, Sun SPARC and UltraSPARC, Motorola 68000, PowerPC, PowerPC64, ARM, Hitachi SuperH, IBM S/390, MIPS, HP PA-RISC, Intel IA-64, DEC VAX, AMD x86-64, AXIS CRIS, Renesas M32R

การเคลื่อนย้ายลินุกซ์ไปยังเครื่องอเนกประสงค์ที่เป็นสถาปัตยกรรม 32 บิต 64 บิต สามารถทำได้โดยง่ายเพราะลินุกซ์มีพีเอ็มเอ็มยู (PMMU : paged memory management unit) และมีช่องทางของจีซีซี (gcc : GNU C compiler) ทำให้ลินุกซ์สามารถย้ายไปบนสถาปัตยกรรมที่ไม่มีพีเอ็มเอ็มยูได้อย่างสะดวกแม้ฟังก์ชันที่ใช้ได้อาจอยู่ในขอบเขตจำกัด

2.2.3 การติดตั้งเคอร์เนล

- ถ้าจะติดตั้งรหัสต้นฉบับเต็มอัตราให้นำเคอร์เนลที่เป็นทาร์บอลไปไว้ในสารบบที่ผู้ติดตั้งได้รับอนุญาตแล้วและทำการขยายออก

```
gzip -cd linux-2.6.XX.tar.gz | tar xvf -
```

หรือ

```
bzip2 -dc linux-2.6.XX.tar.bz2 | tar xvf -
```

โดยแทน XX ด้วยหมายเลขรุ่นและไม่ควรใช้พื้นที่ในสารบบ /usr/src/linux เพราะสารบบนี้มักจะมีเซตหัวเรื่องเคอร์เนลที่ใช้โดยแฟ้มหัวเรื่องคลังโปรแกรมซึ่งอาจจะเกิดความสับสนหากถูกรบกวนจากการขยายททาร์บอล

- ถ้าจะยกระดับระหว่างฉบับที่ 2.6.xx โดยการปะก็ให้นำแฟ้มที่จะปะทั้งหมดเข้าไปที่สารบบระดับสูงสุดของเคอร์เนล ต้นฉบับ (linux-2.6.xx) และดำเนินงานดังนี้

```
gzip -cd ../patch-2.6.xx.gz | patch -p1
```

หรือ

```
bzip2 -dc ../patch-2.6.xx.bz2 | patch -p1
```

โดยแทน xx ด้วยหมายเลขรุ่นที่มากกว่าหมายเลขรุ่นที่ใช้งานอยู่ แต่การยกระดับโดยการปะก็สามารถทำได้อีกวิธีคือใช้บทปะเคอร์เนล (script patch-kernel) โดยบทนี้จะตรวจรุ่นของเคอร์เนลที่ใช้ในปัจจุบันและทำการยกระดับเคอร์เนลด้วยตัวปะที่เหมาะสม

```
linux/scripts/patch-kernel linux
```

โดยอาร์กิวเมนต์แรกของคำสั่งคือที่อยู่ของต้นฉบับเคอร์เนลและการปะจะกระทำการจากสารบบปัจจุบันหากต้องการกระทำการจากสารบบอื่นให้ระบุเป็นอาร์กิวเมนต์ที่ 2 ของคำสั่ง

- ถ้าจะยกระดับระหว่างฉบับที่ 2.6.xx.y ด้วยตัวปะรุ่นเสถียร (patch-2.6.xx.y) ให้เข้าใจว่าตัวปะเหล่านี้ไม่ได้เขียนให้ปะเพิ่มจากตัวปะรุ่นก่อนหน้าแต่อย่างใดแต่เขียนมาให้ปะกับฐานของฉบับ 2.6.xx เช่นต้องการยกระดับ 2.6.12 ด้วยตัวปะ 2.6.12.3 ก็ให้ปะ 2.6.12.3 ได้โดยไม่ต้องปะ 2.6.12.1 และ 2.6.12.2 แต่หากต้องการยกระดับ 2.6.12.2 ไปเป็น 2.6.12.3 จะต้องผันกลับตัวปะ 2.6.12.2 (patch -R) ก่อนที่จะปะ 2.6.12.3

- เพื่อให้แน่ใจว่าไม่มีแฟ้ม .o ที่เก่าและใช้งานไม่ได้แล้วและแฟ้มที่มีอยู่ก็มีความน่าเชื่อถือ

```
cd linux
make mrproper
```

ณ จุดนี้จะได้ต้นฉบับที่ถูกต้องที่พร้อมติดตั้ง

2.2.4 ความต้องการทางซอฟต์แวร์

การแปลโปรแกรมและการดำเนินงานเคอร์เนล 2.6.xx ต้องใช้โปรแกรมสำเร็จหลายโปรแกรมที่ใช้ในปัจจุบันโดยดูได้ใน Documentation/Changes สำหรับหมายเลขรุ่นต่ำสุดที่สามารถใช้ได้เพราะการใช้หมายเลขรุ่นที่ไม่เหมาะสมอาจเกิดข้อผิดพลาดที่ยากที่จะติดตาม

2.2.5 สร้างสารบบสำหรับเคอร์เนล

เมื่อทำการแปลโปรแกรมแล้วแฟ้มเอาต์พุตทั้งหมดจะถูกเก็บในรหัสต้นฉบับเคอร์เนลโดยปริยายแต่หากต้องการเก็บแฟ้มเอาต์พุตไปยังสถานที่อื่นก็ต้องระบุผ่านตัวเลือก make O=output/dir เช่น

```
kernel source code:     /usr/src/linux-2.6.N
build directory:        /home/name/build/kernel
```

เพื่อที่จะกำหนดโครงแบบและสร้างเคอร์เนลทำได้ดังนี้

```
cd /usr/src/linux-2.6.N
make O=/home/name/build/kernel menuconfig
make O=/home/name/build/kernel
sudo make O=/home/name/build/kernel modules_install install
```

2.2.6 กำหนดโครงแบบเคอร์เนล

เป็นขั้นตอนที่สำคัญและไม่ควรมองข้ามเพราะเคอร์เนลแต่ละฉบับที่ถูกปล่อยออกมามักจะมีตัวเลือกใหม่ ๆ เพิ่มเข้ามาอยู่เสมอ การกำหนดแฟ้มโครงแบบที่ไม่เหมาะสมอาจเกิดปัญหาที่คาดไม่ถึง ถ้าหากต้องการกำหนดโครงแบบโดยใช้โครงแบบเดิมที่ใช้อยู่ก็ทำได้โดย

```
make oldconfig
```

ซึ่งคำสั่งนี้จะถามเฉพาะคำถามใหม่ที่ไม่มีคำตอบอยู่ในโครงแบบเดิม ส่วนคำสั่งกำหนดโครงแบบชนิดอื่นได้แก่

- make menuconfig เป็นข้อความบนฐานรายการเลือกแบบสี ปุ่มแสดงรายการและคำตอบโต้
- make xconfig เป็นเอกซ์วินโดวส์ (คิวที) บนฐานของเครื่องมือกำหนดโครงแบบ
- make gconfig เป็นเอกซ์วินโดวส์ (จีทีเค) บนฐานของเครื่องมือกำหนดโครงแบบ
- make oldconfig ใช้โครงแบบเดิมที่มีอยู่ในแฟ้ม ./.config

- make silentoldconfig เหมือน make oldconfig ต่างกันเพียงจะไม่แสดงคำถามที่ตอบไปแล้วบนจอมอนิเตอร์

หมายเหตุ สำหรับ make config มีดังต่อไปนี้

- การมีโปรแกรมขับที่ไม่จำเป็นจะทำให้เคอร์เนลมีขนาดใหญ่ขึ้นและมีโอกาสนำไปสู่ปัญหาต่าง ๆ เช่นการพยายามหาแผ่นวงจรตัวควบคุมที่ไม่มีอยู่จริงในระบบอาจส่งผลให้ตัวควบคุมอื่นที่มีอยู่จริงในระบบเกิดความสับสนตามไปด้วย

- การแปลโปรแกรมโดยเซตตัวประมวลผลให้สูงกว่า 386 จะทำให้เคอร์เนลที่ได้ไม่สามารถทำงานบนเครื่อง 386 ได้ ซึ่งเคอร์เนลจะตรวจสอบตัวประมวลผลในระหว่างการปลุกเครื่อง

- ระบบที่มีตัวประมวลผลร่วมคณิตศาสตร์ (Math coprocessor) รวมอยู่ในระบบแล้วการคำนวณต่าง ๆ ของระบบจะทำโดยตัวประมวลผลร่วมคณิตศาสตร์นี้แม้ว่าเคอร์เนลที่แปลโปรแกรมมาจะมีตัวเลียนแบบคณิตศาสตร์ (math-emulation) รวมอยู่ในเคอร์เนลแล้วก็ตาม ข้อดีของเคอร์เนลที่มีตัวเลียนแบบคณิตศาสตร์ก็คือเคอร์เนลนั้นสามารถทำงานบนเครื่องใด ๆ ก็ได้โดยไม่ต้องสนใจว่าเครื่องนั้นมีตัวประมวลผลร่วมคณิตศาสตร์อยู่ในเครื่องหรือไม่แม้ว่าเคอร์เนลที่มีตัวเลียนแบบคณิตศาสตร์รวมอยู่จะมีขนาดใหญ่ขึ้นเล็กน้อยก็ตาม

- โดยปกติการแฮ็กเคอร์เนลจะมีผลทำให้เคอร์เนลมีขนาดใหญ่ขึ้นหรือไม่ก็เคอร์เนลทำงานได้ช้าลง (หรือทั้ง 2 อย่าง) ตลอดจนทำให้เสถียรภาพของเคอร์เนลลดต่ำลง

2.2.7 **แปลโปรแกรมเคอร์เนล**

1. ตรวจสอบรุ่นจีซีซี (gcc) ที่ต้องใช้แปลโปรแกรม

2. สำรองเคอร์เนลรวมถึงมอดูลเดิมไว้ในกรณีมีความผิดพลาดโดยเฉพาะระหว่างการพัฒนาที่แก้จุดบกพร่องไม่ได้

3. สั่ง make เพื่อสร้างภาพเคอร์เนลที่บีบอัด

4. สั่ง make install เพื่อติดตั้งเคอร์เนล

5. สั่ง make modules_install เพื่อติดตั้งมอดูล

6. สำเนาภาพเคอร์เนลไปอยู่ในตำแหน่งที่โปรแกรมบรรจุการปลุกเครื่องมองเห็นหรือดัดแปรโปรแกรมบรรจุการปลุกเครื่องให้มองเห็นภาพเคอร์เนล

2.3 การขัดจังหวะและเอกซ์เซปชัน (3, 4)

2.3.1 บทนำการขัดจังหวะและเอกซ์เซปชัน

การขัดจังหวะถูกนิยามว่าเป็นเหตุการณ์ที่เปลี่ยนลำดับกระทำการคำสั่งโดยหน่วยประมวลผลซึ่งเหตุการณ์ดังกล่าวจะตรงกับสัญญาณไฟฟ้าที่ก่อกำเนิดมาจากวงจรฮาร์ดแวร์ทั้งจากภายในและภายนอกชิปซีพียู โดยการขัดจังหวะมักถูกแบ่งออกเป็นการขัดจังหวะแบบซิงโครนัสและการขัดจังหวะแบบอะซิงโครนัส

- การขัดจังหวะแบบซิงโครนัสจะถูกสร้างจากหน่วยควบคุมของซีพียูขณะที่กำลังกระทำการคำสั่งและสาเหตุที่เรียกซิงโครนัสก็เพราะว่าหน่วยควบคุมจะส่งการขัดจังหวะก็ต่อเมื่อเลิกกระทำการคำสั่งแล้วเท่านั้น

- การขัดจังหวะแบบอะซิงโครนัสจะถูกสร้างโดยอุปกรณ์ฮาร์ดแวร์อื่น ณ เวลาใด ๆ ก็ได้ภายใต้สัญญาณนาฬิกาของซีพียู

ไมโครโพรเซสเซอร์ตระกูลอินเทล 80x86 จะเรียกการขัดจังหวะแบบซิงโครนัสและการขัดจังหวะแบบอะซิงโครนัสว่า “เอกซ์เซปชัน” (Exception) และ “การขัดจังหวะ” (Interrupt) ตามลำดับ ซึ่งในที่นี้ก็จะขอใช้การเรียกดังกล่าวในเอกสารนี้ต่อไป แต่บางครั้งการใช้คำว่า “สัญญาณขัดจังหวะ” อาจจะหมายถึงการขัดจังหวะทั้ง 2 แบบคือแบบซิงโครนัสและแบบอะซิงโครนัส

การขัดจังหวะอาจถูกสร้างหรือปล่อยมาจากอุปกรณ์ไอ/โอ ยกตัวอย่างเช่นการกดแป้นพิมพ์ของผู้ใช้แป้นพิมพ์ทำให้มีสัญญาณขัดจังหวะเกิดขึ้นซึ่งตรงข้ามกับเอกซ์เซปชันที่เกิดจากทั้งความผิดพลาดของโปรแกรมหรือมีเงื่อนไขผิดปรกติที่ต้องจัดกระทำด้วยเคอร์เนลโดยในกรณีแรกเคอร์เนลจะจัดกระทำกับเอกซ์เซปชันด้วยการส่งสัญญาณใดสัญญาณหนึ่งที่เหมาะสมไปให้กระบวนการปัจจุบันที่กำลังกระทำการอยู่ส่วนในกรณีที่สองเคอร์เนลจะแสดงขั้นตอนที่จำเป็นทั้งหมดเพื่อกู้เงื่อนไขที่ผิดปรกติเช่น page fault

2.3.2 บทบาทของสัญญาณขัดจังหวะ

สัญญาณขัดจังหวะถูกใช้เพื่อให้หน่วยประมวลผลเปลี่ยนไปกระทำการคำสั่งอื่นนอกคำสั่งที่กำลังกระทำการอยู่ในปัจจุบัน เมื่อสัญญาณขัดจังหวะมาถึงซีพียูแล้วซีพียูต้องหยุดการทำงานที่กระทำอยู่เพื่อสลับไปทำกิจกรรมใหม่ โดยจะต้องเก็บค่าต่าง ๆ ของโปรแกรมนับเช่นค่า eip และ cs ไปไว้ในกองซ้อนของภาวะเคอร์เนลหลังจากนั้นนำตำแหน่งเลขที่อยู่ที่เกี่ยวข้องกับการขัดจังหวะชนิดนั้น ๆ ไปใส่ในโปรแกรมนับ

การสลับไปจัดกระทำการขัดจังหวะแตกต่างกับการสลับกระบวนการตรงที่การสลับกระบวนการจะมีการสลับบริบทแต่การสลับไปจัดกระทำการขัดจังหวะจะดำเนินงานบนกระบวนการที่กำลังดำเนินงานอยู่ในขณะที่มีการขัดจังหวะเกิดขึ้น

2.3.3 **การขัดจังหวะและเอกซ์เซปชัน**

2.3.3.1 **การขัดจังหวะ**

อินเทลได้แบ่งประเภทของการขัดจังหวะการขัดจังหวะออกเป็น 2 ประเภทได้แก่

1. การขัดจังหวะที่สามารถพรางได้ (Maskable interrupt) จะส่งการขัดจังหวะไปที่พิน INTR ของไมโครโพรเซสเซอร์ซึ่งสามารถปิดทางส่งได้ด้วยการเซตตัวบ่งชี้ IF ของเรจิสเตอร์ eflags ให้อยู่ในสถานะว่างและไออาร์คิวทุกตัวที่ส่งออกจากอุปกรณ์ไอ/โอจะถือเป็นสัญญาณขัดจังหวะที่สามารถพรางได้

2. การขัดจังหวะที่ไม่สามารถพรางได้ (Nonmaskable interrupt) จะส่งการขัดจังหวะไปที่พิน NMI ของไมโครโพรเซสเซอร์ซึ่งการเซตตัวบ่งชี้ IF ของเรจิสเตอร์ eflags ให้อยู่ในสถานะว่างไม่สามารถปิดทางสัญญาณชนิดนี้ได้ สัญญาณนี้เกิดได้จากเหตุการณ์วิกฤตเช่นกรณีมีความขัดข้องทางฮาร์ดแวร์

2.3.3.2 **เอกซ์เซปชัน**

อินเทลได้แบ่งประเภทของเอกซ์เซปชันออกเป็น 2 ประเภทได้แก่

1. เอกซ์เซปชันแบบหน่วยประมวลผลตรวจพบ (Processor-detected exceptions) เกิดเมื่อซีพียูตรวจพบเงื่อนไขผิดปรกติขณะกระทำการตามคำสั่งซึ่งแบ่งย่อยออกไปได้ 3 ชนิดตามค่าเรจิสเตอร์ eip ที่เก็บอยู่ในกองซ้อนของภาวะเคอร์เนลเมื่อหน่วยควบคุมของซีพียูยกเอกซ์เซปชัน

1.1 ความผิดพร่อง (Faults) เกิดเมื่อค่า eip คือเลขที่อยู่ของคำสั่งที่เป็นสาเหตุของความผิดพร่องหลังจากนั้นคำสั่งดังกล่าวสามารถดำเนินต่อไปเมื่อเลิกจัดกระทำเอกซ์เซปชันซึ่งการกลับมาทำคำสั่งเดิมจะทำได้ก็ต่อเมื่อตัวจัดกระทำสามารถแก้เงื่อนไขผิดปรกติที่เป็นสาเหตุของเอกซ์เซปชันได้แล้ว

1.2 กับดัก (Traps) เกิดเมื่อค่า eip คือเลขที่อยู่ของคำสั่งที่จะกระทำการหลังจากที่ได้กระทำการผ่านไปแล้วซึ่งกับดักจะเกิดเมื่อระบบไม่ต้องการกระทำการซ้ำกับคำสั่งที่เลิกหรือสิ้นสุดไปแล้ว วัตถุประสงค์หลักของการใช้กับดักคือใช้เพื่อแก้จุดบกพร่องโดยสัญญาณขัดจังหวะจะแจ้งไปยังตัวแก้จุดบกพร่องว่าคำสั่งที่ระบุได้กระทำการไปแล้ว เมื่อผู้ใช้ตรวจสอบข้อมูลจากตัวแก้จุดบกพร่องเสร็จผู้ใช้สามารถบอกให้โปรแกรมแก้จุดบกพร่องเริ่มกระทำการคำสั่งถัดไปได้ทันที

1.3 เลิกกลางคัน (Aborts) เกิดเมื่อมีความผิดพลาดอย่างรุนแรงจนเป็นเหตุให้หน่วยควบคุมมีปัญหาจนถึงขั้นไม่สามารถเก็บค่าจำเป็นต่าง ๆ ในเรจิสเตอร์ eip เลิกกลางคันอาจมีสาเหตุมาจากความขัดข้องทางฮาร์ดแวร์หรืออาจมาจากค่าที่ไม่สมเหตุสมผลในตารางระบบ สัญญาณขัดจังหวะจะถูกหน่วยควบคุมส่งเป็นสัญญาณฉุกเฉินเพื่อไปสลับการควบคุมให้จัดกระทำเอกซ์เซปชันเลิกกลางคันทันทีโดยการจัดกระทำชนิดนี้จะไม่มีทางเลือกอื่นนอกจากบังคับให้กระบวนการเลิกหรือสิ้นสุดลง

2. เอกซ์เซปชันแบบโปรแกรม (Programmed exceptions) เกิดเมื่อถูกผู้สร้างโปรแกรมร้องขอโดยตรงด้วยคำสั่ง int หรือ int3 จึงมักถูกเรียกว่าเป็น “การขัดจังหวะซอฟต์แวร์” (software interrupts) ส่วนคำสั่ง into และ bound ก็สามารถทำให้เกิดเอกซ์เซปชันแบบโปรแกรมได้ในกรณีที่ทดสอบเงื่อนไขแล้วเป็นเท็จ เอกซ์เซปชันแบบโปรแกรมจะถูกจัดกระทำแบบกับดักจากหน่วยควบคุม วัตถุประสงค์การใช้งานเอกซ์เซปชันชนิดนี้คือใช้สร้างการเรียกระบบหรือใช้แจ้งเหตุการณ์ที่ระบุให้ตัวแก้จุดบกพร่องทราบ

2.3.4 เวกเตอร์ของการขัดจังหวะและเอกซ์เซปชัน

การขัดจังหวะและเอกซ์เซปชันจะถูกระบุด้วยหมายเลขโดยมีพิสัยจาก 0 ถึง 255 โดยอินเทลจะเรียกหมายเลข 8 บิตชนิดไม่มีเครื่องหมายนี้ว่า “เวกเตอร์” ซึ่งเวกเตอร์ของการขัดจังหวะที่ไม่สามารถพรางได้และเอกซ์เซปชันจะถูกตรึงไว้แต่เวกเตอร์ของการขัดจังหวะที่สามารถพรางได้นั้นสามารถเปลี่ยนแปลงได้โดยการโปรแกรมตัวควบคุมการขัดจังหวะ โดยลินุกซ์จะกำหนดเวกเตอร์ให้เป็นดังนี้

- เวกเตอร์พิสัยจาก 0 ถึง 31 จะตรงกับเอกซ์เซปชันและการขัดจังหวะที่ไม่สามารถพรางได้
- เวกเตอร์พิสัยจาก 32 ถึง 47 จะถูกกำหนดให้กับการขัดจังหวะที่สามารถพรางได้หรือก็คือการขัดจังหวะจากไออาร์คิว
- เวกเตอร์ที่เหลือที่มีพิสัยจาก 48 ถึง 255 อาจใช้ระบุการขัดจังหวะทางซอฟทต์แวร์ ซึ่งลินุกซ์จะใช้เวกเตอร์ในส่วนนี้เพียงเวกเตอร์เดียวคือเวกเตอร์ 128 หรือ 0x80 เพื่อใช้สำหรับการเรียกระบบ โดยเมื่อคำสั่งแอสเซมบลี int 0x80 ถูกกระทำการโดยกระบวนการในภาวะผู้ใช้ ซีพียูจะสลับไปยังภาวะเคอร์เนลและเริ่มกระทำการเคอร์เนลฟังก์ชัน system_call()

2.3.5 ไออาร์คิวและการขัดจังหวะ

ตัวควบคุมอุปกรณ์ฮาร์ดแวร์แต่ละตัวสามารถส่งสัญญาณขอขัดจังหวะไปตามสายเอาต์พุตที่เรียกว่า “ไออาร์คิว” สายไออาร์คิวที่มีอยู่ทุกสายจะเชื่อมต่อไปยังอินพุตพินของวงจรฮาร์ดแวร์ที่เรียกว่า “ตัวควบคุมการขัดจังหวะ” โดยตัวควบคุมการขัดจังหวะมีการทำงานดังนี้

1. เฝ้าสังเกตสายไออาร์คิวและตรวจสอบสัญญาณยก

2. ถ้ามีสัญญาณยกเกิดขึ้นบนสายไออาร์คิว

2.1 เปลี่ยนสัญญาณยกที่ได้รับไปสู่เวกเตอร์ที่ตรงกับสัญญาณ

2.2 เก็บเวกเตอร์ในช่องทางไอ/โอของตัวควบคุมการขัดจังหวะ ซึ่งซีพียูสามารถอ่านเวกเตอร์นี้ผ่านทางบัสข้อมูลได้

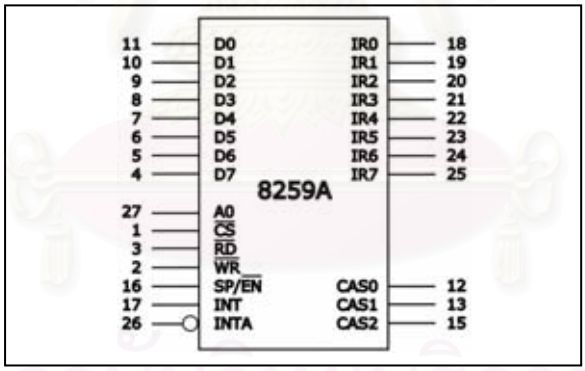
2.3 ส่งสัญญาณยกไปยังพิน INTR ของหน่วยประมวลผล

2.4 รอจนกว่าซีพียูจะตอบรับสัญญาณขัดจังหวะ ซึ่งซีพียูจะตอบรับโดยการเขียนไปยังช่องทางไอ/โอของพีไอซีหรือตัวควบคุมการขัดจังหวะแบบโปรแกรมได้ (PIC : Programable interrupt controller) หลังจากซีพียูตอบรับแล้วสาย INTR จะถูกทำให้อยู่ในสถานะว่าง

3. กลับไปทำข้อ 1

สายไออาร์คิวจะมีลำดับหมายเลขเริ่มตั้งแต่ 0 โดยสายไออาร์คิวแรกมักกำหนดเป็นไออาร์คิว 0 แต่ค่าโดยปริยายของเวกเตอร์ในอินเทลที่สอดคล้องกับไออาร์คิว n คือ n+32 การส่งระหว่างไออาร์คิวและเวกเตอร์สามารถทำได้โดยการส่งคำสั่งไอ/โอที่เหมาะสมไปยังช่องทางตัวควบคุมการขัดจังหวะ

2.3.6 ตัวควบคุมการขัดจังหวะแบบโปรแกรมได้ 8259เอ



รูป 2.1 พินขาออกของพีไอซี 8259เอ

D0 - D7 “การเชื่อมต่อข้อมูล” ใช้เชื่อมต่อข้อมูลแบบสองทางโดยต่อไปยังบัสข้อมูลบนไมโครโพรเซสเซอร์

IR0 - IR7 “อินพุตการขอขัดจังหวะ” ใช้ขอขัดจังหวะและอาจใช้เชื่อมไปยังชิปลูกข่าย 8259เอ ในระบบหลายชิป

$\overline{WR}$ “อินพุตเขียน” ใช้เชื่อมต่อไปยังสัญญาณ $\overline{IOWC}$ บนไมโครโพรเซสเซอร์

$\overline{RD}$ “อินพุตอ่าน” ใช้เชื่อมต่อไปยังสัญญาณ IORC

INT “เอาต์พุตการขัดจังหวะ” ใช้เชื่อมต่อจากชิปหลัก 8259เอ ไปยังพิน INTR บนไมโครโพรเซสเซอร์

INTA “ตอบรับการขัดจังหวะ” ใช้เป็นอินพุตรอการตอบรับขอขัดจังหวะโดยเชื่อมไปยังสัญญาณ INTA บนระบบในกรณีระบบหลายชิปจะมีเพียง $\overline{\text{INTA}}$ ของชิปหลักเท่านั้นที่ใช้เชื่อมต่อ

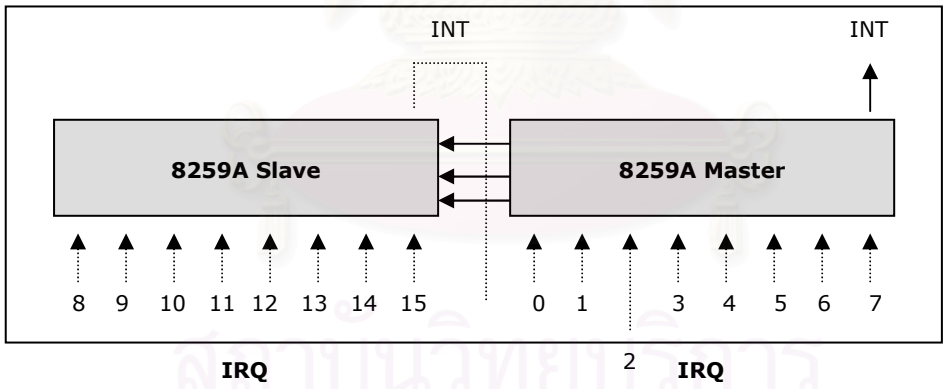
A0 “อินพุตเลขที่อยู่ A0” ใช้เลือกคำของคำสั่งงานที่แตกต่างกันภายในชิป 8259เอ

$\overline{\text{CS}}$ “เลือกชิป” ใช้เปิดทางให้สามารถโปรแกรมและควบคุมชิป 8259เอ ได้

SP/EN “โปรแกรมลูกข่าย/เปิดทางบัฟเฟอร์” ใช้งานได้ 2 ฟังก์ชันตามภาวะบัฟเฟอร์ กรณีอยู่ในภาวะบัฟเฟอร์จะใช้เป็นเอาต์พุตเพื่อควบคุมการรับส่งบัสข้อมูลส่วนกรณีที่ไม่อยู่ในภาวะบัฟเฟอร์จะใช้โปรแกรมอุปกรณ์ว่าเป็นอุปกรณ์หลัก (1) หรือเป็นอุปกรณ์ลูกข่าย (0)

CAS0 - CAS2 “สายต่อเรียง” ใช้เป็นเอาต์พุตเพื่อเชื่อมต่อจากชิปหลักไปยังชิปลูกข่ายในระบบหลายชิป

รูป 2.2 แสดงการเชื่อมอินเทลพีไอซี 8259เอ แบบต่อเรียงที่สามารถจัดกระทำสายอินพุตไออาร์คิวที่แตกต่างกันได้ถึง 15 สายโดยสายเอาต์พุต INT ของพีไอซีตัวที่สองจะเชื่อมต่อไปยังไออาร์คิว 2 ของพีไอซีตัวแรก สัญญาณบนไออาร์คิว 2 แสดงว่ามีสัญญาณใดสัญญาณหนึ่งเกิดขึ้นบนไออาร์คิว 8 ถึงไออาร์คิว 15 หมายเลขไออาร์คิวสำหรับการต่อแบบนี้จะถูกจำกัดที่ 15 หมายเลขเท่านั้น



รูป 2.2 การเชื่อมต่อพีไอซี 8259เอ จำนวน 2 ตัวแบบต่อเรียง

เมื่อหมายเลขสายไออาร์คิวมีจำกัดจึงมีความจำเป็นที่อุปกรณ์ไอ/โอที่แตกต่างกันหลาย ๆ อุปกรณ์ต้องใช้สายไออาร์คิวร่วมกัน เมื่อเป็นเช่นนี้อุปกรณ์ทั้งหมดที่ใช้ไออาร์คิวร่วมสายกันจะต้องถูกหยั่งสัญญาณตามลำดับโดยซอฟต์แวร์จัดกระทำสัญญาณเพื่อตัดสินว่าอุปกรณ์ใดเป็นผู้ส่งสัญญาณขอขัดจังหวะ

สายไออาร์คิวแต่ละสายสามารถเลือกที่จะปิดทางได้โดยการโปรแกรมให้พีไอซีปิดทางไออาร์คิวที่ต้องการหรือก็คือพีไอซีสามารถบอกให้หยุดส่งการขัดจังหวะที่ถูกโปรแกรมให้ปิดทางและในทางตรงข้ามก็สามารถโปรแกรมให้พีไอซีเปิดทางไออาร์คิวที่ต้องการได้เช่นกัน การขัดจังหวะที่

ถูกปิดทางจะไม่มีการสูญหายแต่อย่างใดโดยพีไอซีจะส่งการขัดจังหวะดังกล่าวให้ซีพียูทันทีที่มีการเปิดทางให้การขัดจังหวะดังกล่าว ลักษณะเช่นนี้นิยมใช้สำหรับจัดกระทำการขัดจังหวะเพราะสามารถประมวลผลไออาร์คิวชนิดเดียวกันแบบเรียงได้

การเลือกเปิดทางหรือปิดทางไออาร์คิวจะไม่เหมือนกับการพรางหรือไม่พรางของการขัดจังหวะที่สามารถพรางได้ เมื่อตัวบ่งชี้ IF ของเรจิสเตอร์ eflags อยู่ในสถานะว่างการส่งการขัดจังหวะที่สามารถพรางได้โดยพีไอซีจะถูกเมินจากซีพียู โดยคำสั่งแอสเซมบลี cli และ sti คือคำสั่งไม่เซตและเซตตัวบ่งชี้ตามลำดับ

2.3.7 **เอกซ์เซปชัน**

อินเทล 80x86 ไมโครโพรเซสเซอร์มีการใช้เอกซ์เซปชันประมาณ 20 เอกซ์เซปชันที่แตกต่างกัน โดยเคอร์เนลจะต้องจัดหาการจัดกระทำให้กับแต่ละเอกซ์เซปชันซึ่งบางเอกซ์เซปชันหน่วยควบคุมของซีพียูจะสร้างรหัสค่าผิดพลาดฮาร์ดแวร์และผลักค่าดังกล่าวไปยังกองซ้อนของภาวะเคอร์เนลก่อนที่จะเริ่มจัดกระทำเอกซ์เซปชัน

ตาราง 2.1 เอกซ์เซปชัน

#	Exception Name	Exception Handler	Signal	Type	Description
0	Divide error	divide_error()	SIGFPE	fault	Raised when program tries to divide by 0
1	Debug	debug()	SIGTRAP	trap or fault	Raised when the T flag of eflag is set or when the address of an instruction or operand fall whitin the range of an active register
2	NMI	nmi()	None	None	Reserve for nonmaskable interrupt
3	Breakpoint	int3()	SIGTRAP	trap	Caused by an int3 instruction
4	Overflow	overflow()	SIGSEGV	trap	An into instruction has been executed when the OF flag of eflag is set
5	Bounds check	bounds()	SIGSEGV	fault	A bound instruction has been executed with the operand outside of the valid address bounds
6	Invalid opcode	invalid_op()	SIGILL	fault	The CPU execution unit ha detected an invalid opcode
7	Device not available	device_not_available()	SIGSEGV	fault	An ESCAPE or MMX instruction has been executed with the TS flag of cr0 set
8	Double fault	double_fault()	SIGSEGV	abort	Normally, when the CPU detects an exception while trying to call the handler for a prior exception, the two exceptions can be handled serially. In a few cases, however, the processor cannot handle them serially, hence it raises this exception
9	Coprocessor segment overrun	coprocessor_segment_ overrun()	SIGFPE	abort	Problems with the external mathematical coprocessor
10	Invalid TSS	invalid_tss()	SIGSEGV	fault	The CPU has attempted a context switch to a process having an invalid Task State Segment
11	Segment not present	segment_not_present()	SIGBUS	fault	A reference was made to a segment not present in memory
12	Stack segment	stack_segment()	SIGBUS	fault	The instruction attempted to exceed the stack segment limit, or the segment identified by ss is not present in memory
13	General protection	general_protection()	SIGSEGV	fault	One of the protection rules in the protected mode of the Intel 80x86 has been violated
14	Page fault	page_fault()	SIGSEGV	fault	The addressed page is not present in memory, the corresponding page table entry is null, or a violation of the paging protection mechanism has occurred
15	Intel reserved	None			These value are reserved by Intel for future development
16	Floating point error	coprocessor_error()	SIGFPE	fault	The floating point unit integrated into the CPU chip has signaled an error condition, such as numeric overflow or division by 0
17	Alignment check	alignment_check()	SIGSEGV	fault	The address of an operand is not correctly aligned
18 to 31	Intel reserved	None	None	None	These value are reserved by Intel for future development

2.4 **สัญญาณ** (3, 4)

2.4.1 **บทนำสัญญาณ**

สัญญาณถูกแนะนำขึ้นมาใช้งานเป็นครั้งแรกในระบบยูนิกซ์เพื่อทำให้การสื่อสารระหว่างกระบวนการสามารถทำได้ง่ายขึ้นโดยเคอร์เนลจะใช้สัญญาณเพื่อบอกหรือแจ้งไปยังกระบวนการให้ทราบว่ามีเหตุการณ์บางอย่างเกิดขึ้นและสัญญาณส่วนใหญ่ในระบบยูนิกซ์สามารถมองเห็นได้จากภาวะผู้ใช้

ระบบยูนิกซ์มีการใช้งานสัญญาณมากกว่า 35 ปีโดยที่มีการเปลี่ยนแปลงน้อยมากเพราะสัญญาณที่ใช้มาตั้งแต่ต้นนั้นใช้งานง่ายและยังมีประสิทธิภาพสูงอีกทั้งยังได้รับความนิยมในการนำไปใช้งานอย่างแพร่หลายจนถึงปัจจุบัน

2.4.2 **บทบาทของสัญญาณ**

สัญญาณคือข่าวสารสั้น ๆ ที่ใช้ส่งไปยังกระบวนการหรือกลุ่มของกระบวนการ ซึ่งปกติข่าวสารที่ส่งไปยังกระบวนการจะเป็นตัวเลขเพื่อระบุสัญญาณโดยที่สัญญาณตามมาตรฐานจะไม่มีพื้นที่สำหรับรับอาร์กิวเมนต์หรือข่าวสารหรือข้อมูลอื่นใดนอกจากตัวเลขที่ระบุสัญญาณ โดยสัญญาณถูกออกแบบมาเพื่อ 2 วัตถุประสงค์ใหญ่ดังนี้

1. เพื่อให้กระบวนการสามารถรับรู้เหตุการณ์ที่ได้ระบุไว้ในทันทีที่มีเหตุการณ์เกิดขึ้น
2. เพื่อให้กระบวนการกระทำการตามค่าโดยปริยายหรือเรียกชุดคำสั่งจัดเรื่องขัดจังหวะภายหลังจากได้รับสัญญาณแล้ว

คุณลักษณะสำคัญของสัญญาณคือสัญญาณสามารถถูกส่งไปยังกระบวนการ ณ เวลาใด ๆ โดยที่ไม่สามารถคาดเดาสถานะของกระบวนการ ณ เวลานั้นได้ ซึ่งสัญญาณที่ถูกส่งไปให้กระบวนการที่ไม่ได้อยู่ในช่วงดำเนินงานตัวสัญญาณนั้นจะถูกเก็บโดยเคอร์เนลจนกว่ากระบวนการนั้นจะกลับมาดำเนินงานเคอร์เนลจึงค่อยส่งสัญญาณที่เก็บให้กระบวนการนั้นต่อไป โดยเคอร์เนลจะแยกการทำงานกับสัญญาณเป็น 2 เฟสคือ

1. เฟสส่งสัญญาณ โดยเคอร์เนลจะปรับตัวบอกของกระบวนการปลายทางเพื่อแสดงว่าสัญญาณใหม่ถูกส่งไปแล้ว
2. เฟสรับสัญญาณ โดยเคอร์เนลจะบังคับให้กระบวนการปลายทางตอบสนองกับสัญญาณด้วยการเปลี่ยนสถานะการดำเนินงานหรือเริ่มการดำเนินงานชุดคำสั่งจัดเรื่องขัดจังหวะตามสัญญาณที่ได้รับ

สัญญาณที่ถูกส่งออกไปแล้วแต่ยังไม่ถึงผู้รับจะถูกเรียกว่า “สัญญาณค้างหรือสัญญาณคอย” (pending signal) ซึ่ง ระยะเวลาที่สัญญาณค้างต้องรอก่อนถึงผู้รับนั้นไม่สามารถระบุเวลาที่แน่นอนได้เพราะขึ้นกับองค์ประกอบดังต่อไปนี้

- สัญญาณจะถูกกระบวนการรับได้ก็ต่อเมื่อกระบวนการนั้นกำลังดำเนินงานอยู่เท่านั้น

- สัญญาณนั้นอาจถูกกระบวนการเลือกที่จะบล็อกซึ่งในกรณีนี้สัญญาณนั้นไม่มีทางที่จะถูกรับจนกว่าการบล็อกนั้นจะมีการแก้ไขให้หายไป

- เมื่อกระบวนการกำลังดำเนินงานชุดคำสั่งจัดเรื่องขัดจังหวะให้สัญญาณใดอยู่กระบวนการนั้นจะอยู่ในสถานะพรางการตอบโต้กับสัญญาณอื่นกล่าวคือกระบวนการจะบล็อกสัญญาณอื่นจนกว่าการดำเนินงานชุดคำสั่งจัดเรื่องขัดจังหวะสัญญาณที่กำลังทำ ณ ขณะนั้นเสร็จสิ้นเพราะการดำเนินงานชุดคำสั่งจัดเรื่องขัดจังหวะสัญญาณไม่สามารถถูกขัดจังหวะจากชุดคำสั่งจัดเรื่องขัดจังหวะสัญญาณอื่นได้และฟังก์ชันนั้นไม่ต้องการกลับเข้าใหม่ (reentrant)

แม้ว่าการใช้งานสัญญาณเป็นสิ่งที่ผู้ใช้สามารถทำความเข้าใจได้ง่ายแต่เบื้องหลังการทำงานของสัญญาณนี้เคอร์เนลจะต้องทำงานที่สลับซับซ้อนหลายอย่างเช่น

- เคอร์เนลจะต้องจำสัญญาณที่แต่ละกระบวนการเลือกที่จะบล็อกได้

- ขณะสลับจากภาวะเคอร์เนลไปภาวะผู้ใช้เคอร์เนลจะต้องตรวจสอบว่ามีสัญญาณใหม่สำหรับแต่ละกระบวนการเข้ามาหรือไม่ซึ่งการตรวจนี้จะเกิดทุก ๆ สัญญาณขัดจังหวะของตัวจับเวลาซึ่งมีค่าประมาณ 10 มิลลิวินาที

- เคอร์เนลต้องตัดสินว่าสัญญาณนั้นสามารถเมินหรือไม่สนใจได้หรือไม่ซึ่งจะเกิดขึ้นได้เมื่อเงื่อนไขทั้ง 3 ต่อไปนี้เป็นจริง

1 กระบวนการปลายทางไม่ถูกติดตามจากกระบวนการอื่น (ตัวบ่งชี้ PF_TRACED ในตัวบอกกระบวนการมีค่าเป็น 0)

2 สัญญาณไม่ถูกเลือกบล็อกจากกระบวนการปลายทาง

3 สัญญาณถูกเมินหรือไม่สนใจจากกระบวนการปลายทางทั้งกรณีถูกเมินหรือไม่สนใจอย่างแจ่มแจ้งหรือตัวกระบวนการไม่เปลี่ยนค่าโดยปริยายที่ถูกตั้งให้เมินหรือไม่สนใจสัญญาณ

- การเรียกชุดคำสั่งจัดเรื่องขัดจังหวะสัญญาณนั้นเคอร์เนลอาจจะต้องมีการสลับกระบวนการไปยังชุดคำสั่งจัดเรื่องขัดจังหวะที่จุดเวลาใด ๆ ขณะที่กระบวนการกำลังดำเนินงานอยู่และต้องสลับบริบทกลับทันทีที่ชุดคำสั่งจัดเรื่องขัดจังหวะรีเทิร์น

สัญญาณที่ใช้ในลินุกซ์อาจรับมาจากหลายทางแตกต่างกันไปเช่นรับมาจาก BSD หรือ System V หรือ POSIX ก็เป็นไปได้ทั้งนั้น

2.4.3 โครงสร้างข้อมูลที่เกี่ยวข้องกับสัญญาณ

โครงสร้างพื้นฐานที่ใช้เก็บสัญญาณที่จะส่งไปให้กระบวนการมีลักษณะเป็นแถวลำดับของบิตที่ใช้ชื่อว่า sigset_t โดยแต่ละบิตจะแทนสัญญาณแต่ละชนิด

```
typedef struct {
        unsigned long sig[2];
} sigset_t;
```

เมื่อแต่ละ unsigned long มีพื้นที่เท่ากับ 32 บิตจำนวนสัญญาณมากสุดที่อาจประกาศใช้ได้ในลินุกซ์จึงเท่ากับ 64 แต่ไม่มีสัญญาณใดที่มีหมายเลขสัญญาณเท่ากับ 0 จึงเหลือบิตสัญญาณเพียง 63 บิตซึ่งแบ่งได้ 2 ส่วนคือ 31 บิตที่อยู่ในส่วนแรกของ sigset_t สามารถดูได้ในตาราง 2.2 ส่วน 32 บิตที่อยู่ในส่วนที่สองของ sigset_t จะถือให้เป็น “สัญญาณเวลาจริงหรือสัญญาณทันที” (real-time signal) โดยตัวบอกกระบวนการที่คอยติดตามการส่งสัญญาณไปให้กระบวนการประกอบไปด้วยเขตข้อมูลดังต่อไปนี้

- signal เป็นตัวแปรชนิด sigset_t ใช้เก็บสัญญาณที่จะส่งให้กระบวนการ
- blocked เป็นตัวแปรชนิด sigset_t ใช้เก็บสัญญาณที่กระบวนการเลือกที่จะบล็อก
- sigpending เป็นตัวบ่งชี้โดยจะอยู่ในสถานะเซตถ้ามีสัญญาณค้างหรือมีสัญญาณคอยอยู่โดยที่สัญญาณนั้นไม่ถูกกระบวนการปลายทางเลือกที่จะบล็อก
- gsig เป็นตัวชี้ไปยังโครงสร้างข้อมูล signal_struct ที่พรรณนาถึงชุดคำสั่งจัดเรื่องขัดจังหวะของสัญญาณชนิดต่าง ๆ โดย signal_struct มีโครงสร้างดังนี้

```
struct signal_struct {
        atomic_t            count;
        struct k_sigacton   action[64];
        spinlock_t          siglock;
};
```

โครงสร้างนี้อาจจะมีกระบวนการหลายกระบวนการมาใช้งานร่วมกันเพื่อลดขนาดโครงสร้างข้อมูลของกระบวนการที่จัดเรื่องขัดจังหวะสัญญาณลงซึ่งถ้าไม่ใช้โครงสร้างร่วมกันแล้วแต่ละกระบวนการต้องใช้พื้นที่เพิ่มขึ้นเพื่อโครงสร้างนี้อีกประมาณ 1,300 ไบต์ต่อ 1 กระบวนการโดยตัวแปร count จะแทนจำนวนกระบวนการที่ใช้โครงสร้าง signal_struct ร่วมกันและตัวแปร action[64] จะแทน 64 โครงสร้าง k_sigaction ที่เก็บชุดคำสั่งจัดเรื่องขัดจังหวะของแต่ละสัญญาณส่วนตัวแปร siglock ใช้เพื่อซิงโครนัสการเข้าสู่เขตข้อมูล

บางสถาปัตยกรรมมีการกำหนดคุณสมบัติของสัญญาณให้มองเห็นได้เฉพาะจากเคอร์เนลเท่านั้นโดยเก็บคุณสมบัติสัญญาณในโครงสร้าง k_sigaction แต่แพลตฟอร์มของอินเทลมีการออกแบบให้คุณสมบัติของสัญญาณสามารถมองเห็นได้จากภาวะผู้ใช้โดยปรับโครงสร้าง k_sigaction มาอยู่ในโครงสร้าง sa ชนิด sigaction แทน

2.4.4 ตัวสัญญาณ

ตาราง 2.2 31 สัญญาณแรกที่รองรับโดยลินุกซ์ 2.2 สำหรับอินเทล 80x86

#	Signal Name	Default Action	Comment	POSIX
1	SIGHUP	Abort	Hangup of controlling terminal or process	Yes
2	SIGINT	Abort	Interrupt from keyboard	Yes
3	SIGQUIT	Dump	Quit from keyboard	Yes
4	SIGILL	Dump	Illegal instruction	Yes
5	SIGTRAP	Dump	Breakpoint for debugging	No
6	SIGABRT	Dump	Abnormal termination	Yes
	SIGIOT	Dump	Equivalent to SIGABRT	No
7	SIGBUS	Abort	Bus error	No
8	SIGFPE	Dump	Floating point exception	Yes
9	SIGKILL	Abort	Forced process termination	Yes
10	SIGUSR1	Abort	Available to processes	Yes
11	SIGSEGV	Dump	Invalid memory reference	Yes
12	SIGUSR2	Abort	Available to processes	Yes
13	SIGPIPE	Abort	Write to pipe with no readers	Yes
14	SIGALRM	Abort	Real timer clock	Yes
15	SIGTERM	Abort	Process termination	Yes
16	SIGSTKFLT	Abort	Coprocessor stack error	No
17	SIGCHLD	Ignore	Child process stopped or terminated	Yes
18	SIGCONT	Continue	Resume execution, if stopped	Yes
19	SIGSTOP	Stop	Stop process execution	Yes
20	SIGTSTP	Stop	Stop process issued from tty	Yes
21	SIGTTIN	Stop	Background process requires input	Yes
22	SIGTTOU	Stop	Background process requires output	Yes
23	SIGURG	Ignore	Urgent condition on socket	No
24	SIGXCPU	Abort	CPU time limit exceeded	No
25	SIGXFSZ	Abort	File size limit exceeded	No
26	SIGVTALRM	Abort	Virtual timer clock	No
27	SIGPROF	Abort	Profile time clock	No
28	SIGWINCH	Ignore	Window resizing	No
29	SIGIO	Abort	I/O now possible	No
	SIGPOLL	Abort	Equivalent to SIGIO	No
30	SIGPWR	Abort	Power supply failure	No
31	SIGUNUSED	Abort	Not used	No

ตัวสัญญาณมีให้ใช้งานหลายมาตรฐานซึ่งผู้ใช้สามารถสังเกตตัวสัญญาณได้จากชื่อเพราะส่วนมากเซตของแมโครที่ขึ้นต้นด้วย SIG จะเป็นชื่อของสัญญาณโดยแต่ละสัญญาณจะมีหมายเลขกำกับเป็นของตัวเองตามมาตรฐานที่แตกต่างกันไปเช่น SIGCHLD จะมีหมายเลขกำกับเท่ากับ 17 ในลินุกซ์โดยจะถูกส่งจากกระบวนการลูกไปให้กระบวนการแม่เพื่อแจ้งให้กระบวนการแม่ทราบว่ากระบวนการลูกเข้าสู่ภาวะหยุดหรือภาวะเลิกแล้วหรือ SIGSEGV ซึ่งมีหมายเลขกำกับเท่ากับ 11 ในลินุกซ์จะถูกส่งให้กระบวนการเมื่อหน่วยความจำที่กระบวนการอ้างถึงไม่มีอยู่จริงหรือไม่สามารถเข้าใช้งานได้เป็นต้น ส่วนในช่อง POSIX ของตาราง 2.2 แสดงถึงการรองรับกับคลาสสัญญาณใหม่ในมาตรฐานโพสิกซ์ที่เรียกว่าสัญญาณทันที

2.4.5 การตอบสนองสัญญาณ

กระบวนการมี 3 ทางเลือกในการตอบสนองสัญญาณ

1. เมินหรือไม่สนใจ (ignore) สัญญาณอย่างแจ่มแจ้ง

2. กระทำการตามค่าโดยปริยาย (default) ที่สอดคล้องกับสัญญาณ (Default action ในตาราง 2.2) ซึ่งค่าโดยปริยายนี้จะถูกกำหนดล่วงหน้าจากเคอร์เนลโดยจะขึ้นอยู่กับชนิดของสัญญาณและค่าโดยปริยายอาจเป็นไปได้จากค่าใดค่าหนึ่งต่อไปนี้

- Abort กระบวนการจะถูกทำลายหรือลบทิ้ง
- Dump กระบวนการจะถูกทำลายหรือลบทิ้งแต่แกนของแฟ้มข้อมูลที่เก็บบริบทของการกระทำการจะถูกสร้างขึ้นอาจเพื่อวัตถุประสงค์ในการนำไปใช้แก้จุดบกพร่อง
- Ignore สัญญาณจะถูกเมินหรือไม่ใส่ใจ
- Stop กระบวนการจะหยุดหรือถูกจัดให้อยู่ในสถานะ TASK_STOPPED
- Continue ถ้ากระบวนการกำลังอยู่ในสถานะ TASK_STOPPED กระบวนการจะถูกจัดไปยังสถานะ TASK_RUNNING

3. จับสัญญาณ (catch) แล้วให้กระบวนการเรียกชุดคำสั่งจัดเรื่องขัดจังหวะสัญญาณที่สอดคล้องกับสัญญาณนั้น ๆ

SIGKILL และ SIGSTOP เป็นสัญญาณที่ไม่สามารถเมินหรือไม่สนใจหรือแม้แต่จะไปเรียกชุดคำสั่งจัดเรื่องขัดจังหวะสัญญาณได้ เมื่อผู้รับได้รับสัญญาณดังกล่าวจะต้องกระทำการตามค่าโดยปริยายทันทีซึ่งก็คือผู้รับจะถูกทำลายหรือลบทิ้งในกรณี SIGKILL และผู้รับจะต้องหยุดในกรณี SIGSTOP แต่จะมีข้อยกเว้นสำหรับ 2 กระบวนการพิเศษคือกระบวนการ 0 และกระบวนการ 1 ที่ไม่ต้องคำนึงถึงการป้องกันสัญญาณจากโปรแกรมที่กำลังกระทำการตัวอย่างเช่นสัญญาณทุกสัญญาณที่ส่งถึงกระบวนการ 0 (swapper) จะถูกละทิ้งทันทีซึ่งทำให้กระบวนการ 0 ไม่มีทางตาย ส่วนสัญญาณที่ส่งถึงกระบวนการ 1 (init) จะถูกละทิ้งในกรณีที่กระบวนการ 1 ไม่ต้องการจับสัญญาณซึ่งทำให้กระบวนการ 1 จะตายได้กรณีเดียวคือโปรแกรม init จบการทำงานแล้วเท่านั้น

2.4.6 สัญญาณทันที

มาตรฐานโพสิกซ์ได้แนะนำคลาสสัญญาณใหม่ที่เรียกว่าสัญญาณทันทีสอดคล้องกับหมายเลข 32 ถึง 63 โดยข้อแตกต่างระหว่างสัญญาณมาตรฐานกับสัญญาณทันทีคือ

- สัญญาณทันทีชนิดเดียวกันสามารถเข้าคิวเพื่อรอการจัดเรื่องขัดจังหวะจากกระบวนการได้ซึ่งทำให้แน่ใจได้ว่าสัญญาณชนิดเดียวกันที่ส่งไปให้กระบวนการจะไม่มีสัญญาณใดถูกละทิ้ง

- มีการกำหนดลำดับความสำคัญให้กับสัญญาณทันทีโดยสัญญาณทันทีที่มีหมายเลขต่ำกว่าจะมีลำดับความสำคัญสูงกว่าทำให้สัญญาณคอยที่มีลำดับความสำคัญสูงกว่าจะถูกส่งออกไปก่อน

โครงสร้างคิวของสัญญาณทันที

```
struct signal_queue {
        struct signal_queue *next;
        siginfo_t info;
};
```

โดยตัวแปร next ชนิด signal_queue จะชี้ไปยังส่วนย่อยถัดไปที่แสดงในรายการ ส่วนตัวแปร info ชนิด siginfo_t จะเก็บสารสนเทศของสัญญาณทันทีที่รับมาและต้องส่งไปให้กระบวนการปลายทางเช่น pid ของผู้ส่งและ uid ของเจ้าของสัญญาณ

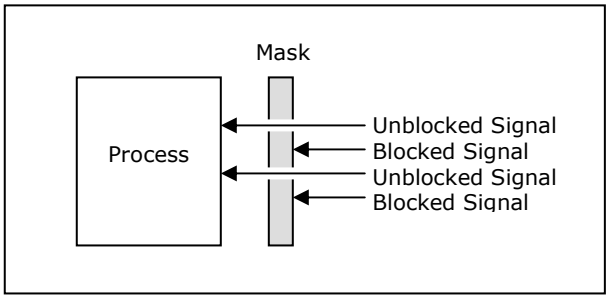
ตัวบอกกระบวนการของแต่ละกระบวนการจะมี 2 เขตข้อมูลคือ sigqueue ที่ชี้ไปยังส่วนย่อยแรกของคิวสัญญาณทันทีและ sigqueue_tail ที่จะชี้ไปยังเขตข้อมูล next ของส่วนย่อยสุดท้ายของคิว

เมื่อมีการส่งสัญญาณทางฟังก์ชัน send_sig_info() จะตรวจสอบว่าหมายเลขสัญญาณที่ส่งมีค่ามากกว่า 31 หรือไม่กรณีที่มีค่ามากกว่า 31 สัญญาณที่ส่งจะถูกแทรกเข้าไปในคิวสัญญาณทันทีของกระบวนการปลายทาง

สัญญาณทันทีที่มีหมายเลขต่ำสุดจะแทนด้วย SIGRTMIN และสัญญาณทันทีที่มีหมายเลขสูงสุดจะแทนด้วย SIGRTMAX การเรียกใช้งานใช้ได้ตั้งแต่ค่าต่ำสุดไปถึงค่าสูงสุดซึ่งมีค่าที่ไม่แน่นอน ตัวอย่างการเรียกใช้งานเช่น SIGRTMIN, SIGRTMIN+1, ... , SIGRTMAX-1, SIGRTMAX

2.4.7 การบล็อกสัญญาณด้วยตัวพราง

กระบวนการสามารถสร้างตัวพรางเพื่อบล็อกสัญญาณที่กระบวนการไม่ต้องการรับหรือใช้เพื่อป้องกันไม่ให้กระบวนการถูกขัดจังหวะจากสัญญาณที่ไม่ต้องการ โดยการบล็อกสัญญาณต่างกับการเมินหรือไม่สนใจตรงที่การเมินหรือไม่สนใจสัญญาณนั้นทางผู้รับได้รับสัญญาณเข้ามาแล้วแต่ไม่มีการกระทำการอะไรกับสัญญาณที่เมินหรือไม่สนใจแตกต่างกับการบล็อกสัญญาณที่ผู้รับจะไม่ได้รับสัญญาณที่เลือกบล็อกจนกว่าผู้รับเลือกที่จะไม่บล็อกสัญญาณนั้นโดยสัญญาณที่ถูกบล็อกจะจัดอยู่ในสถานะของสัญญาณคอย



รูป 2.3 การบล็อกสัญญาณด้วยตัวพราง

2.4.8 การใช้งานหรือควบคุมสัญญาณในระดับกระบวนการ

การใช้งานสัญญาณนอกจากจะต้องรู้จักกับตัวสัญญาณแล้วผู้ใช้จำเป็นจะต้องทราบวิธีจัดการหรือควบคุมสัญญาณต่าง ๆ ให้ทำงานได้ตามความต้องการซึ่งฟังก์ชันที่เกี่ยวข้องกับการบังคับใช้สัญญาณมีอยู่หลายฟังก์ชันซึ่งในที่นี้จะยกตัวอย่างฟังก์ชันและวิธีใช้อย่างย่อที่เห็นว่าสำคัญและจำเป็นสำหรับการเริ่มต้นที่จะเรียนรู้การใช้สัญญาณเช่นการส่ง การรับ การบล็อก ฯลฯ ซึ่งผู้ที่เข้าใจฟังก์ชันเบื้องต้นเหล่านี้ดีแล้วและต้องการใช้งานฟังก์ชันอื่นเพิ่มเติมก็สามารถศึกษาได้เองโดยง่าย

ตาราง 2.3 ตัวอย่างฟังก์ชันที่ใช้จัดการกับสัญญาณ

System Call	Description
kill()	Send a signal to a process
sigaction()	Change the action associated with a signal
signal()	Similar to sigaction()
sigpending()	Check whether there are pending signals
sigprocmask()	Modify the set of blocked signals
sigsuspend()	Wait for a signal
rt_sigaction	Change the action associated with a real-time signal
rt_sigpending()	Check whether there are pending real-time signals
rt_sigprocmask()	Modify the set of blocked real-time signals
rt_sigqueueinfo()	Send a real-time signal to a process
rt_sigsuspend()	Wait for a real-time signal
rt_sigtimewait()	Similar to rt_sigsuspend()

2.4.8.1 ฟังก์ชัน kill()

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int signum);
```

ใช้ส่งสัญญาณไปให้กระบวนการหรือกลุ่มกระบวนการ โดยฟังก์ชันนี้ต้องการ 2 อาร์กิวเมนต์คือ

1. pid เพื่อระบุไอดีของกระบวนการที่ต้องการส่งสัญญาณไปให้โดย

1.1 ถ้า pid > 0 สัญญาณ sig จะถูกส่งไปกระบวนการที่มีไอดีของกระบวนการเท่ากับ pid

1.2 ถ้า pid = 0 สัญญาณ sig จะถูกส่งไปทุก ๆ กระบวนการที่อยู่ภายในกลุ่มกระบวนการเดียวกับกระบวนการที่เรียกฟังก์ชันนี้

1.3 ถ้า pid = -1 สัญญาณ sig จะถูกส่งไปทุก ๆ กระบวนการยกเว้นกระบวนการ 0 กระบวนการ 1 และกระบวนการที่เรียกฟังก์ชันนี้

1.4 ถ้า pid < -1 สัญญาณ sig จะถูกส่งไปทุก ๆ กระบวนการที่อยู่ภายในกลุ่มกระบวนการเดียวกับกระบวนการที่มีไอดีของกระบวนการเท่ากับ -pid

2. signum เพื่อระบุสัญญาณที่ต้องการส่งไปให้กระบวนการ แต่ถ้า signum มีค่าเป็น 0 จะไม่มีการส่งสัญญาณ

ฟังก์ชัน kill() จะรีเทิร์น 0 ในกรณีส่งสัญญาณออกไปได้สำเร็จและจะรีเทิร์น -1 ในกรณีเกิดความผิดพลาด

2.4.8.2 **ฟังก์ชัน** sigaction()

```
#include <signal.h>
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

ใช้เปลี่ยนการกระทำของกระบวนการให้สอดคล้องกับสัญญาณที่ได้รับ ซึ่งการกระทำจะอยู่ในสถานะเซตไปจนกว่าจะมีการเปลี่ยนแปลงเป็นอย่างอื่น โดยฟังก์ชันนี้ต้องการ 3 อาร์กิวเมนต์คือ

1. signum เพื่อระบุสัญญาณสำหรับการกระทำที่กำหนด ยกเว้น SIGKILL กับ SIGSTOP

2. act เพื่อระบุการกระทำที่ต้องการให้กระบวนการทำเมื่อกระบวนการได้รับสัญญาณ signum

3. oldact เพื่อใช้เก็บการกระทำก่อนหน้าการกระทำที่ใช้อยู่ในปัจจุบัน 1 ขั้น

โครงสร้าง sigaction ถูกกำหนดไว้ดังนี้

```
struct sigaction {
        void (*sa_handler)(int);                    /* POSIX 1003.1   signal handler */
        void (*sa_sigaction)(int, siginfo_t *, void *)          /* POSIX 1003.1b realtime signal
handler */
        sigset_t sa_mask;
        int sa_flags;
        void (*sa_restorer)(void);
}
```

1. sa_handler และ sa_sigaction เพื่อระบุชนิดของการกระทำที่จะทำโดยค่านี้สามารถระบุเป็นตัวชี้ให้ชี้ไปยังชุดคำสั่งจัดเรื่องขัดจังหวะสัญญาณหรือระบุเป็น SIG_DFL เพื่อกระทำการตามค่าโดยปริยายหรือระบุเป็น SIG_IGN เพื่อเมินหรือไม่สนใจสัญญาณอย่างแจ่มแจ้ง

2. sa_mask เพื่อระบุตัวพรางสัญญาณที่กระบวนการต้องการบล็อก นอกจากนั้นสัญญาณที่กำลังอยู่ระหว่างการดำเนินการชุดคำสั่งจัดเรื่องขัดจังหวะจะถูกบล็อกโดยปริยายถ้าไม่มีการระบุตัวบ่งชี้ SA_NODEFER หรือ SA_NOMASK

3. sa_flags เพื่อระบุเซตของตัวบ่งชี้ที่ใช้ดัดแปรพฤติกรรมของกระบวนการจัดเรื่องขัดจังหวะสัญญาณ ซึ่งทำขึ้นโดยการออร์ 0 กับตัวบ่งชี้ที่ต้องการโดยตัวอย่างของตัวบ่งชี้ได้แก่

3.1 SA_NOCLDSTOP ถ้า signum คือ SIGCHLD กระบวนการที่ระบุตัวบอกนี้ใน sa_flags จะไม่ได้รับการแจ้งเมื่อกระบวนการลูกหยุดตัวอย่างเช่นเมื่อกระบวนการลูกได้รับ SIGSTOP, SIGTSTP, SIGTTIN หรือ SIGTTOU

3.2 SA_ONESHOT หรือ SA_RESETHAND กระบวนการที่ระบุตัวบอกนี้ใน sa_flags เมื่อได้รับสัญญาณจะทำงานตามชุดคำสั่งจัดเรื่องขัดจังหวะที่กำหนดไว้เพียงครั้งเดียวเท่านั้นก่อนที่การกระทำครั้งต่อ ๆ ไปหลังได้รับสัญญาณจะกระทำการตามค่าโดยปริยาย

3.3 SA_RESTART กระบวนการที่ระบุตัวบอกนี้ใน sa_flags จะทำให้การเรียกระบบที่ถูกขัดจังหวะจากชุดคำสั่งจัดเรื่องขัดจังหวะสัญญาณสามารถเริ่มทำต่อได้

3.4 SA_NOMASK หรือ SA_NODEFER กระบวนการที่ระบุตัวบอกนี้ใน sa_flags จะไม่บล็อกสัญญาณชนิดเดียวกันกับสัญญาณที่ชุดคำสั่งจัดเรื่องขัดจังหวะกำลังดำเนินงานอยู่ในปัจจุบัน

3.5 SA_SIGINFO กระบวนการที่ระบุตัวบอกนี้ใน sa_flags จะใช้ sa_sigaction แทน sa_handler

4. sa_restorer ไม่ค่อยมีผู้ใช้งานแล้วและไม่มีการรองรับในมาตรฐานโพสิกซ์

ฟังก์ชัน sigaction() จะรีเทิร์น 0 ในกรณีทำงานสำเร็จและจะรีเทิร์น -1 ในกรณีเกิดความผิดพลาด

2.4.8.3 **ฟังก์ชัน** sigprocmask(), sigpending(), sigsuspend()

```
#include <signal.h>
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
int sigpending(sigset_t *set);
int sigsuspend(const sigset_t *mask);
```

ฟังก์ชัน sigprocmask() ใช้ดัดแปรตัวพรางสัญญาณของกระบวนการซึ่งแต่ละกระบวนการจะมีตัวพรางสัญญาณเป็นของตัวเอง โดยฟังก์ชันนี้ต้องการ 3 อาร์กิวเมนต์คือ

1. how เพื่อระบุวิธีที่จะใช้กับอาร์กิวเมนต์ set ซึ่งกำหนดได้ 3 แบบดังนี้

1.1 SIG_BLOCK เพื่อให้กระบวนการบล็อกสัญญาณที่ระบุในตัวแปร set โดยการเพิ่มเข้ากับสัญญาณบล็อกเดิม

1.2 SIG_UNBLOCK เพื่อให้กระบวนการไม่บล็อกสัญญาณที่ระบุในตัวแปร set โดยการลบออกจากสัญญาณบล็อกเดิม

1.3 SIG_SETMASK เพื่อให้กระบวนการบล็อกสัญญาณที่กำหนดในตัวแปร set โดยใช้แทนสัญญาณบล็อกเดิมทั้งหมด

2. set เพื่อระบุชุดของสัญญาณที่จะดัดแปรตัวพรางสัญญาณ

3. oldset เพื่อระบุที่เก็บชุดสัญญาณก่อนหน้าชุดสัญญาณ set ที่ใช้อยู่ในปัจจุบัน 1 ขั้น

ฟังก์ชัน sigpending() ใช้ตรวจสอบการมีอยู่ของสัญญาณคอยโดยนำไปเก็บที่ตัวแปร set

ฟังก์ชัน sigsuspend() กระบวนการที่เรียกฟังก์ชันนี้จะใช้ชุดสัญญาณที่อยู่ในตัวแปร mask แทนตัวพรางสัญญาณปัจจุบันชั่วคราวโดยกระบวนการที่เรียกฟังก์ชันนี้จะค้างหรือคอยไปจนกว่าจะได้รับสัญญาณ

ฟังก์ชัน sigprocmask(), sigpending(), sigsuspend() จะรีเทิร์น 0 ในกรณีทำงานสำเร็จ และจะรีเทิร์น -1 ในกรณีเกิดความผิดพลาด

2.4.8.4 **ฟังก์ชัน** sigemptyset(), sigfillset(), sigaddset(), sigdelset(), sigismember()

```
#include <signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
int sigismember(const sigset_t *set, int signum);
```

ใช้ดัดแปรหรือตรวจสอบ sigset_t โดย

ฟังก์ชัน sigemptyset() ใช้กำหนดสถานะว่างหรือไม่เซตให้กับบิตทุกบิตที่อยู่ในตัวแปร set

ฟังก์ชัน sigfillset() ใช้กำหนดสถานะเซตให้กับบิตทุกบิตที่อยู่ในตัวแปร set

ฟังก์ชัน sigaddset() ใช้เพิ่มสัญญาณที่มีหมายเลข signum เข้าไปในชุดสัญญาณที่อยู่ในตัวแปร set

ฟังก์ชัน sigdelset() ใช้ลบสัญญาณที่มีหมายเลข signum ออกจากชุดสัญญาณที่อยู่ในตัวแปร set

ฟังก์ชัน sigismember() ใช้ทดสอบว่าสัญญาณที่มีหมายเลข signum เป็นสมาชิกของชุดสัญญาณที่อยู่ในตัวแปร set หรือไม่

ฟังก์ชัน sigemptyset(), sigfillset(), sigaddset(), sigdelset() จะรีเทิร์น 0 ในกรณีสำเร็จและจะรีเทิร์น -1 ในกรณีเกิดความผิดพลาด

ฟังก์ชัน sigismember() จะรีเทิร์น 1 ในกรณี signum เป็นสมาชิกของชุดสัญญาณที่อยู่ในตัวแปร set หรืออยู่ในสถานะเซตและจะรีเทิร์น 0 ถ้าไม่ได้เป็นสมาชิกและจะรีเทิร์น -1 ในกรณีเกิดความผิดพลาด

2.4.8.5 **ฟังก์ชัน** rt_*

ฟังก์ชันที่กล่าวข้างต้นเป็นตัวอย่างฟังก์ชันที่ใช้กับสัญญาณมาตรฐานเท่านั้นส่วนฟังก์ชันที่ใช้กับสัญญาณทันทีจะขึ้นต้นด้วย rt_ (ดูตาราง 2.3) ซึ่งมีลักษณะการใช้งานที่ใกล้เคียงกับฟังก์ชันสำหรับสัญญาณมาตรฐานที่ได้กล่าวไปแล้วและจะไม่ขอกล่าวถึงรายละเอียดซ้ำอีก

2.4.9 **การใช้งานหรือควบคุมสัญญาณในระดับสายโยงใย**

การใช้สัญญาณในระดับสายโยงใยมีข้อแตกต่างไปจากการใช้สัญญาณในระดับกระบวนการเพียงเล็กน้อยเท่านั้น ฟังก์ชันที่ใช้ส่วนใหญ่ก็เป็นฟังก์ชันที่กล่าวมาแล้วในการใช้สัญญาณในระดับกระบวนการแต่จะมีฟังก์ชันเพิ่มเติมอีกเล็กน้อยที่จะกล่าวถึงในที่นี้เพื่อให้สามารถใช้สัญญาณในระดับสายโยงใยได้

2.4.9.1 **ฟังก์ชัน** pthread_kill()

```
#include <pthread.h>
#include <signal.h>
int pthread_kill(pthread_t thread, int signum);
```

ใช้ส่งสัญญาณไปยังสายโยงใยที่ต้องการ โดยฟังก์ชันนี้ต้องการ 2 อาร์กิวเมนต์คือ

1. thread เพื่อระบุสายโยงใยที่ต้องการส่งสัญญาณไปให้
2. signum เพื่อระบุหมายเลขของสัญญาณที่ต้องการส่ง

ฟังก์ชัน pthread_kill() จะรีเทิร์น 0 ในกรณีส่งสัญญาณได้สำเร็จหรือรีเทิร์นตัวเลขอื่นเพื่อบอกความผิดพลาดที่เกิดขึ้น

2.4.9.2 **ฟังก์ชัน** pthread_sigmask()

```
#include <pthread.h>
#include <signal.h>
int pthread_sigmask(int how, const sigset_t  *set,  sigset_t  *oldset);
```

ใช้ดัดแปรตัวพรางสัญญาณของสายโยงใยซึ่งแต่ละสายโยงใยจะมีตัวพรางสัญญาณเป็นของตัวเองโดยสายโยงใยที่ถูกสร้างขึ้นใหม่จะใช้ตัวพรางสัญญาณเหมือนกับตัวพรางสัญญาณของผู้สร้างสายโยงใยนั้น ๆ โดยฟังก์ชันนี้ต้องการ 4 อาร์กิวเมนต์คือ

1. how เพื่อระบุวิธีที่จะใช้กับอาร์กิวเมนต์ set ซึ่งกำหนดได้ 3 แบบดังนี้

1.1 SIG_BLOCK เพื่อให้สายโยงใยบล็อกสัญญาณที่ระบุในตัวแปร set โดยเพิ่มเข้าไปในสัญญาณบล็อกเดิม

1.2 SIG_UNBLOCK เพื่อให้สายโยงใยไม่บล็อกสัญญาณที่กำหนดในตัวแปร set โดยลบออกจากสัญญาณบล็อกเดิม

1.3 SIG_SETMASK เพื่อให้สายโยงใยบล็อกสัญญาณที่กำหนดในตัวแปร set โดยใช้แทนสัญญาณบล็อกเดิมทั้งหมด

2. set เพื่อระบุชุดของสัญญาณที่จะดัดแปรตัวพรางสัญญาณ

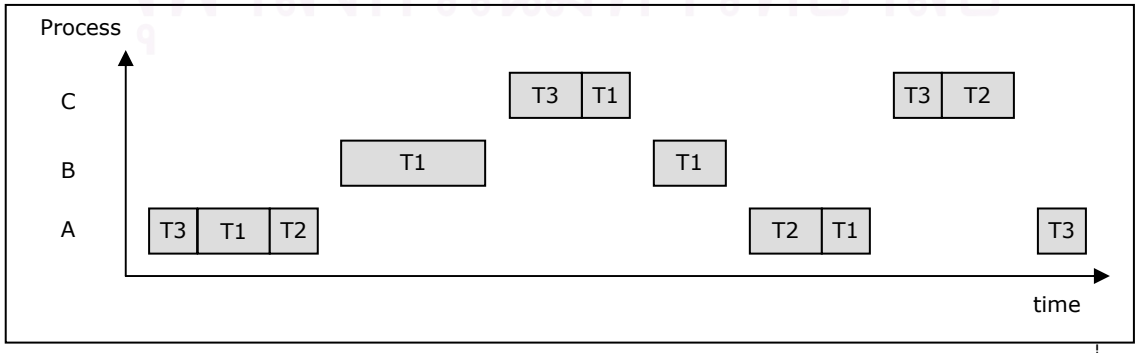
3. oldset เพื่อระบุที่เก็บตัวพรางสัญญาณเดิมหรือในกรณีที่ไม่ต้องการเก็บตัวพรางสัญญาณเดิมให้แทนด้วย NULL

ฟังก์ชัน pthread_sigmask() จะรีเทิร์น 0 ในกรณีทำงานสำเร็จหรือรีเทิร์นตัวเลขอื่นเพื่อบอกความผิดพลาดที่เกิดขึ้น

2.5 **สายโยงใย** (5)

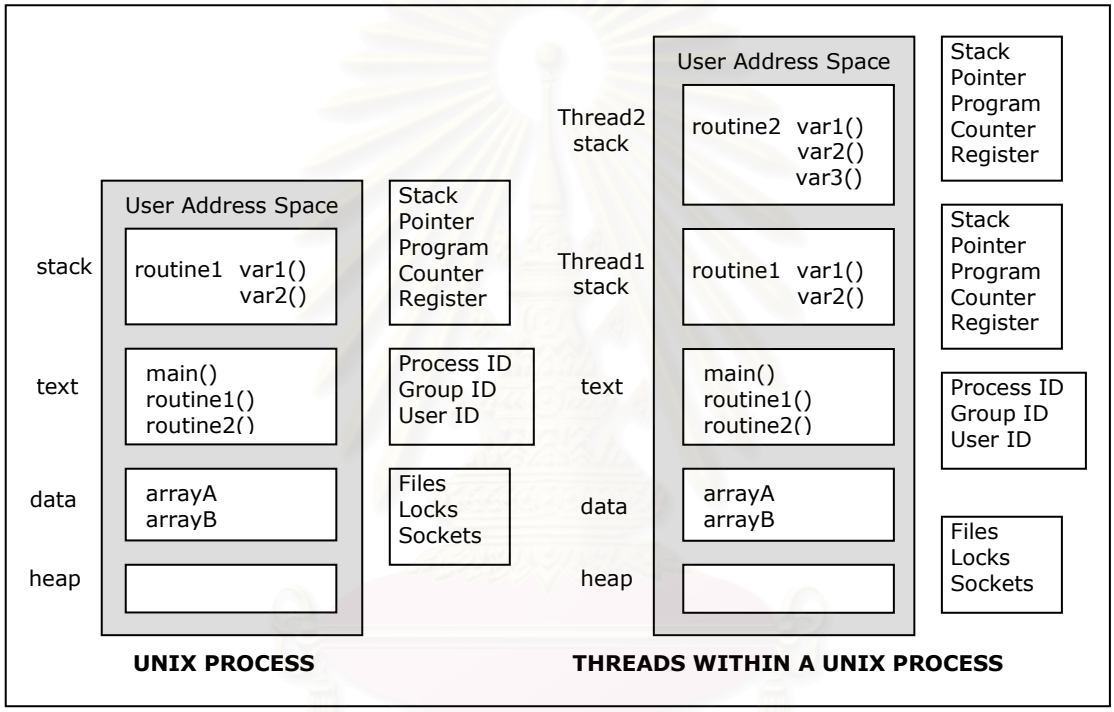
2.5.1 **บทนำสายโยงใย**

คือสายงานหนึ่งของกระบวนการซึ่งแต่ละกระบวนการสามารถสร้างสายโยงใยขึ้นได้หลายสายโดยแต่ละสายจะทำงานสลับกันไปตามการจัดตารางเวลาของระบบปฏิบัติการ



รูป 2.4 ตัวอย่างการสลับเวลาทำงานของสายโยงใยและกระบวนการในระบบหน่วยประมวลผลเดี่ยว

เมื่อก่อนผู้ผลิตฮาร์ดแวร์จะสร้างหรือกำหนดกรรมวิธีในการใช้สายโยงใยให้เป็นไปตามฮาร์ดแวร์ของผู้ผลิตเองการจะนำโปรแกรมที่มีสายโยงใยสำหรับฮาร์ดแวร์หนึ่งไปใช้กับฮาร์ดแวร์จากผู้ผลิตอื่นอาจมีปัญหาเกิดขึ้นได้ ระบบยูนิกซ์จึงกำหนดมาตรฐานการเขียนโปรแกรมสายโยงใยด้วยภาษาซีขึ้นมาในปี ค.ศ.1995 และเรียกว่ามาตรฐาน “IEEE POSIX 1003.1c” ซึ่งเป็นที่มาของชื่อ “POSIX threads” หรือเรียกย่อว่า “Pthreads” โดยกระบวนการและฟังก์ชันที่สามารถเรียกใช้ได้มักจะกำหนดอยู่ใน pthread.h กับ libc และในปัจจุบันผู้ผลิตฮาร์ดแวร์ส่วนใหญ่จะผลิตฮาร์ดแวร์ที่สามารถรองรับ IEEE POSIX 1003.1c เป็นมาตรฐานเอพีไอสำหรับการเขียนโปรแกรมที่มีการใช้สายโยงใย



รูป 2.5 กระบวนการของยูนิกซ์และสายโยงใยในกระบวนการของยูนิกซ์

ดังที่กล่าวในตอนต้นว่าสายโยงใยเป็นส่วนหนึ่งของกระบวนการดังนั้นการมีอยู่หรือการทำงานของสายโยงใยจะต้องอาศัยทรัพยากรของกระบวนการบางส่วนและมีทรัพยากรของสายโยงใยเองอีกบางส่วนทำให้อาจมองได้ว่าการทำงานแบบสายโยงใยก็คือการทำงานแบบกึ่งกระบวนการนั้นเอง

ตัวอย่างของสิ่งที่สายโยงใยต้องใช้ร่วมกับกระบวนการและสายโยงใยอื่นที่อยู่บนกระบวนการเดียวกันได้แก่

- คำสั่งกระบวนการ
- ข้อมูลส่วนใหญ่
- ตัวบอก
- สัญญาณและตัวจัดกระทำสัญญาณ

- สารบบที่ทำงานในปัจจุบัน
- ไอดีผู้ใช้และไอดีกลุ่มและไอดีกระบวนการ
- ฮีป

ตัวอย่างของสิ่งที่สายโยงใยต้องมีเป็นของตัวเองได้แก่

- ไอดีของสายโยงใย
- กลุ่มของเรจิสเตอร์
- ตัวชี้สแต็ก
- ตัวแปรท้องถิ่นภายในสแต็ก
- เลขที่อยู่รีเทิร์น
- ตัวพารามิเตอร์
- ลำดับชั้น
- ค่ารีเทิร์นผิดพลาด

2.5.2 เปรียบเทียบข้อดีข้อด้อยระหว่างสายโยงใยและกระบวนการ

1. ทั้งสายโยงใยและกระบวนการต่างมีข้อดีในการทำงานด้วยการแบ่งแยกงานออกเป็นส่วน ๆ แล้วให้แต่ละส่วนสามารถทำงานสลับกันไปแบบขนานซึ่งลักษณะเช่นนี้เหมาะกับงานที่ต้องทำหน้าที่หลายอย่างที่แตกต่างกันหรือมีอิสระจากกันเช่นให้สายโยงใยหรือกระบวนการหนึ่งทำงานประมวลผลข้อมูลและให้สายโยงใยหรือกระบวนการอีกส่วนหนึ่งรอรับข้อมูลจากช่องทางไอ/โอที่มักเป็นแบบอะซิงโครนัสโดยเมื่อมีข้อมูลเข้ามาก็ส่งสัญญาณขัดจังหวะให้ซีพียูทราบเพื่อทำการรับข้อมูลเป็นต้น ซึ่งโปรแกรมที่ใช้สายโยงใยหรือกระบวนการตั้งแต่ 2 สายหรือ 2 กระบวนการขึ้นไปจะทำงานได้โดดเด่นมากเมื่อทำงานอยู่บนระบบหน่วยประมวลผลแบบหลายหน่วยโดยการทำงานของสายโยงใยหรือกระบวนการจะถูกแยกไปทำงานบนหน่วยประมวลผลแต่ละตัวพร้อม ๆ กันตามการจัดตารางเวลาของระบบปฏิบัติการทำให้การทำงานเป็นแบบขนานอย่างชัดเจน

2. สายโยงใยแต่ละสายภายในกระบวนการเดียวกันสามารถใช้ข้อมูลหรือตัวแปรครอบคลุม (Global variable) ร่วมกันได้ง่ายและเร็วกว่าการสื่อสารระหว่างกระบวนการที่จะต้องใช้การส่งข้อมูลหรือไอพีซี (IPC : Inter Process Communication) แต่เนื่องจากสายโยงใยแต่ละสายที่อยู่ในกระบวนการเดียวกันมีการใช้หน่วยความจำร่วมกันถ้าหากข้อมูลในหน่วยความจำถูกสายโยงใยใดสายโยงใยหนึ่งทำให้ข้อมูลเกิดความผิดพลาดหรือเสียหายแล้วสายโยงใยอื่นที่ต้องใช้ข้อมูลเหล่านั้นก็จะได้รับผลกระทบไปด้วยในขณะที่กระบวนการจะใช้หน่วยความจำต่างพื้นที่กันความผิดพลาดของข้อมูลในกระบวนการหนึ่งจะไม่กระทบกับข้อมูลของกระบวนการอื่นถ้าไม่มีการส่งข้อมูลที่ผิดพลาดนั้นทางไอพีซี

3. สายโยงใยมีข้อจำกัดที่ต้องทำงานบนเครื่องเพียง 1 เครื่องเท่านั้นในขณะที่กระบวนการสามารถทำงานบนเครื่องหลาย ๆ เครื่องพร้อมกันได้

4. ระยะเวลาและทรัพยากรที่ใช้ในการสร้างสายโยงใยจากฟังก์ชัน pthread_create() เฉลี่ยแล้วน้อยกว่าระยะเวลาและทรัพยากรที่ใช้ในการสร้างกระบวนการจากฟังก์ชัน fork()

5. การเปลี่ยนการทำงานจากสายโยงใยหนึ่งไปอีกสายโยงใยหนึ่งบนกระบวนการเดียวกันสามารถทำได้เร็วกว่าการเปลี่ยนการทำงานจากกระบวนการหนึ่งไปอีกกระบวนการหนึ่ง

6. การเขียนโปรแกรมโดยใช้สายโยงใยจะประหยัดหน่วยความจำได้ดีกว่าการเขียนโปรแกรมแบบแยกกระบวนการเพราะแต่ละสายโยงใยที่สร้างจากกระบวนการเดียวกันจะใช้ปริภูมิเลขที่อยู่ (address space) อันเดียวกัน

7. สายโยงใยจะจบการทำงานทันทีที่กระบวนการแม่ (parent process) ที่สร้างสายโยงใยนั้น ๆ จบการทำงานขณะที่การเขียนโปรแกรมแบบหลายกระบวนการเมื่อมีกระบวนการใดจบการทำงานแล้วกระบวนการอื่นก็ยังสามารถทำงานต่อไปได้

2.5.3 **ตัวอย่างงานที่เหมาะสมกับการใช้สายโยงใย**

1. งานที่สามารถแบ่งออกเป็นสายงานย่อยและให้ทำงานขนานกับงานอื่นได้

2. งานที่ต้องรองรับกับเหตุการณ์ประเภทอะซิงโครนัสหรืองานที่ต้องรอเหตุการณ์บางอย่างเป็นเวลานาน ๆ

2.5.4 **ตัวอย่างฟังก์ชันในมาตรฐาน** IEEE POSIX 1003.1c

IEEE POSIX 1003.1c มีฟังก์ชันให้เลือกใช้งานอยู่หลายฟังก์ชันโดยในที่นี้จะขอยกตัวอย่างเฉพาะฟังก์ชันที่สำคัญและจำเป็นในการสร้างและควบคุมสายโยงใยให้ทำงานได้ในระดับพื้นฐานซึ่งเหมาะสำหรับการเริ่มต้นการใช้งานสายโยงใยเท่านั้นสำหรับผู้ที่เข้าใจฟังก์ชันพื้นฐานเหล่านี้ดีแล้วและต้องการใช้ฟังก์ชันอื่นเพิ่มเติมก็สามารถศึกษาทำความเข้าใจต่อได้เองโดยง่าย

2.5.4.1 **ฟังก์ชัน** pthread_create()

```
#include <pthread.h>
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine, void*),
                   void *arg);
```

ใช้สร้างสายโยงใยขึ้นใหม่ โดยฟังก์ชันนี้ต้องการ 4 อาร์กิวเมนต์คือ

1. thread เพื่อใช้เก็บไอดีสายโยงใยใหม่ในกรณีเรียกฟังก์ชันสำเร็จ

2. attr เพื่อระบุแอ็ตทริบิวที่จะใช้หรือให้แทนด้วย NULL ในกรณีต้องการใช้ค่าโดยปริยาย

3. start_routine เพื่อระบุฟังก์ชันที่ต้องการให้สายโยงใยใหม่กระทำการ

4. arg เพื่อระบุอาร์กิวเมนต์ที่ต้องการส่งไปยังฟังก์ชันของสายโยงใยใหม่ ในกรณีต้องการส่งอาร์กิวเมนต์หลายตัวให้ส่งเป็นโครงสร้างของอาร์กิวเมนต์แทน

ฟังก์ชัน pthread_create() จะรีเทิร์น 0 ในกรณีสร้างสายโยงใยใหม่สำเร็จหรือรีเทิร์นตัวเลขอื่นเพื่อบอกความผิดพลาดที่เกิด

2.5.4.2 **ฟังก์ชัน** pthread_self()

```
#include <pthread.h>
pthread_t pthread_self(void);
```

ใช้เรียกไอดีสายโยงใยของสายโยงใยที่เรียกฟังก์ชันนี้ โดยไม่ต้องการอาร์กิวเมนต์ใด ๆ

ฟังก์ชัน pthread_self() จะรีเทิร์นไอดีสายโยงใยของสายโยงใยที่เรียกฟังก์ชันนี้ส่วนการรีเทิร์นความผิดพลาดยังไม่มีการกำหนด

2.5.4.3 **ฟังก์ชัน** pthread_join()

```
#include <pthread.h>
int pthread_join(pthread_t thread, void **status);
```

ใช้บล็อกการทำงานของผู้เรียกฟังก์ชัน pthread_join() จนกว่าสายโยงใยที่ระบุในฟังก์ชันจะทำงานเสร็จ โดยฟังก์ชันนี้ต้องการ 2 อาร์กิวเมนต์คือ

1. thread เพื่อระบุสายโยงใยที่ต้องการรอ

2. status เพื่อใช้เก็บสถานภาพที่รีเทิร์นจาก pthread_exit() ของสายโยงใยที่ระบุในฟังก์ชันในกรณีที่สถานภาพนั้นไม่เป็น NULL

ฟังก์ชัน pthread_join() จะรีเทิร์น 0 ในกรณีทำงานสำเร็จหรือรีเทิร์นตัวเลขอื่นเพื่อบอกความผิดพลาดที่เกิด

2.5.4.4 **ฟังก์ชัน** pthread_mutex_lock()

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

ใช้ยึดมิวเท็กซ์ (mutex) ในกรณีที่มิวเท็กซ์นั้นถูกยึดอยู่แล้วสายโยงใยที่เรียกฟังก์ชันนี้จะถูกบล็อกจนกว่ามิวเท็กซ์นั้นจะถูกปล่อย โดยฟังก์ชันนี้ต้องการอาร์กิวเมนต์ 1 อาร์กิวเมนต์คือ

1. mutex เพื่อระบุมิวเท็กซ์ที่ต้องการยึด

ฟังก์ชัน pthread_mutex_lock() จะรีเทิร์น 0 ในกรณีทำงานสำเร็จหรือรีเทิร์นตัวเลขอื่นเพื่อบอกความผิดพลาดที่เกิดขึ้น

2.5.4.5 **ฟังก์ชัน** pthread_mutex_trylock()

```
#include <pthread.h>
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

ใช้ยึดมิวเท็กซ์ในกรณีที่มิวเท็กซ์นั้นถูกยึดอยู่แล้วฟังก์ชันจะรีเทิร์น EBUSY ทันที โดยฟังก์ชันนี้ต้องการอาร์กิวเมนต์ 1 อาร์กิวเมนต์คือ

1. mutex เพื่อระบุมิวเท็กซ์ที่ต้องการยึด

ฟังก์ชัน pthread_mutex_trylock() จะรีเทิร์น 0 ในกรณีล็อกและสามารถยึดมิวเท็กซ์มาได้สำเร็จหรือรีเทิร์นตัวเลขอื่นเพื่อบอกความผิดพลาดที่เกิดขึ้น

2.5.4.6 **ฟังก์ชัน** pthread_mutex_unlock()

```
#include <pthread.h>
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

ใช้ปล่อยมิวเท็กซ์ที่ได้ยึดไว้ โดยฟังก์ชันนี้ต้องการอาร์กิวเมนต์ 1 อาร์กิวเมนต์คือ

1. mutex เพื่อระบุมิวเท็กซ์ที่ต้องการปล่อย

ฟังก์ชัน pthread_mutex_unlock() จะรีเทิร์น 0 ในกรณีทำงานสำเร็จหรือรีเทิร์นตัวเลขอื่นเพื่อบอกความผิดพลาดที่เกิดขึ้น

2.5.4.7 **ฟังก์ชัน** pthread_cond_wait()

```
#include <pthread.h>
int pthread_cond_wait(pthead_cond_t *cond, pthread_mutex_t *mutex);
```

ใช้รอเงื่อนไข cond โดยปล่อยมิวเท็กซ์ที่ระบุในฟังก์ชันที่ยึดไว้ชั่วคราวและทำการบล็อกการทำงานของสายโยงใยที่เรียกฟังก์ชันนี้จนกว่าเงื่อนไข cond ที่รอจะรีเทิร์นสำเร็จหลังจากนั้นผู้เรียกจะยึดมิวเท็กซ์ที่ปล่อยไปชั่วคราวกลับมาไว้ดังเดิม โดยฟังก์ชันนี้ต้องการ 2 อาร์กิวเมนต์คือ

1. cond เพื่อระบุเงื่อนไขการรอ

2. mutex ระบุมิวเท็กซ์ที่ต้องการปล่อยชั่วคราว

ฟังก์ชัน pthread_cond_wait() จะรีเทิร์น 0 ในกรณีทำงานสำเร็จหรือรีเทิร์นตัวเลขอื่นเพื่อบอกความผิดพลาดที่เกิดขึ้น

2.5.4.8 **ฟังก์ชัน** pthread_cond_signal()

```
#include <pthread.h>
int pthread_cond_signal(pthread_cond_t *cond);
```

ใช้ปลดบล็อกสายโยงใยที่ถูกบล็อกโดยตัวแปร cond อย่างน้อย 1 สายโดยฟังก์ชันนี้ต้องการ 1 อาร์กิวเมนต์คือ

1. cond เพื่อระบุเงื่อนไขที่จะปลดบล็อกให้กับสายโยงใยที่ถูกบล็อกโดยตัวแปร cond

ฟังก์ชัน pthread_cond_signal() จะรีเทิร์น 0 ในกรณีทำงานสำเร็จหรือรีเทิร์นตัวเลขอื่นเพื่อบอกความผิดพลาดที่เกิดขึ้น

2.5.4.9 **ฟังก์ชัน** pthread_cond_broadcast()

```
#include <pthread.h>
int pthread_cond_broadcast(pthread_cond_t *cond);
```

ใช้ปลดบล็อกสายโยงใยทุกสายที่ถูกบล็อกโดยตัวแปร cond โดยฟังก์ชันนี้ต้องการ 1 อาร์กิวเมนต์

1. cond เพื่อระบุเงื่อนไขที่จะปลดบล็อกให้กับสายโยงใยที่ถูกบล็อกโดยตัวแปร cond

ฟังก์ชัน pthread_cond_broadcast() จะรีเทิร์น 0 ในกรณีทำงานสำเร็จหรือรีเทิร์นตัวเลขอื่นเพื่อบอกความผิดพลาดที่เกิดขึ้น

2.5.4.10 **ฟังก์ชัน** pthread_cancel()

```
#include <pthread.h>
int pthread_cancel(pthread_t target_thread);
```

ใช้ยกเลิกการทำงานของสายโยงใยซึ่งปกติการยกเลิกจะประวิงเวลาไปถึงจุดยกเลิกของสายโยงใยนั้นจึงจะสามารถทำการยกเลิกได้ โดยฟังก์ชันนี้ต้องการ 1 อาร์กิวเมนต์คือ

1. target_thread เพื่อระบุสายโยงใยเป้าหมายที่ต้องการให้ยกเลิกการทำงาน

ฟังก์ชัน pthread_cancel() จะรีเทิร์น 0 ในกรณีทำงานสำเร็จหรือรีเทิร์นตัวเลขอื่นเพื่อบอกความผิดพลาด

ตัวอย่างของจุดยกเลิกได้แก่ pthread_join(), pthread_cond_wait(), sem_wait(), sigwait(), pthread_cond_timedwait(), thread_testcancel() เป็นต้นโดยสายโยงใยที่ทำงาน

มาถึงจุดยกเลิกแล้วมักจะต้องตรวจว่ามีคำสั่งยกเลิกสายโยงใยตนเองรอการทำงานอยู่หรือไม่ถ้ามีคำสั่งยกเลิกรออยู่การยกเลิกก็จะเกิดขึ้นที่จุดนี้ทันที

2.5.4.11 **ฟังก์ชัน** pthread_exit()

```
#include <pthread.h>
void pthread_exit(void *value_ptr);
```

ใช้ออกจากสายโยงใยหรือทำให้สายโยงใยนั้นสิ้นสุดการทำงานลงโดยฟังก์ชันนี้ต้องการ 1 อาร์กิวเมนต์คือ

1. value_ptr เพื่อระบุสภานภาพการออกในกรณีที่สายโยงใยไม่เป็นแบบดีแทช (detach) หรือแทนด้วย NULL ถ้าต้องการใช้ค่าโดยปริยาย

ฟังก์ชัน pthread_exit() จะไม่มีการรีเทิร์นค่าให้ผู้ที่เรียกฟังก์ชันนี้

2.5.5 **ปัญหาที่มักพบในการเขียนโปรแกรมที่ใช้สายโยงใย**

การเขียนโปรแกรมที่มีสายโยงใยตั้งแต่ 2 สายขึ้นไปมักจะมีการติดต่อสื่อสารหรือมีการรับส่งข้อมูลระหว่างสายโยงใยซึ่งอาจมีปัญหาเกิดขึ้นได้ถ้าเขียนโปรแกรมไม่รัดกุมพอโดยปัญหาที่พบบ่อยในการเขียนโปรแกรมที่มีการใช้สายโยงใยตั้งแต่ 2 สายขึ้นไปได้แก่ปัญหาติดตายและปัญหาเงื่อนไขแข่งขัน

1. ติดตาย (deadlock) คือสภาวะที่สิ่ง 2 สิ่งหรือมากกว่าเกิดการติดขัดไม่สามารถทำงานหรือดำเนินงานต่อไปได้ ซึ่งในกรณีสายโยงใยอาจเกิดจากสายโยงใย 2 สายต่างต้องการยึดมิวเท็กซ์ที่โดนอีกสายโยงใยหนึ่งยึดไว้แล้ว

2. เงื่อนไขแข่งขัน (race condition) มาจากสภาวะที่สิ่ง 2 สิ่งหรือมากกว่ากำลังแข่งขันทำงานของตนเองให้เสร็จโดยในขณะแข่งขันนั้นผู้แข่งขันต่างต้องใช้บางสิ่งบางอย่างร่วมกันและลำดับการใช้ก่อนหลังหรือการใช้พร้อมกันมีผลให้ผลลัพธ์ที่ได้มีความหลากหลายแตกต่างกัน

<table>
<tr><th colspan="2">Without Mutex</th><th colspan="2">With Mutex</th></tr>
<tr><td colspan="2">int counter=0;

/* Function C */
void functionC() {

  counter++

}</td><td colspan="2">/* Note scope of variable and mutex are the same */
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter=0;
/* Function C */
void functionC() {
  pthread_mutex_lock(&mutex1);
  counter++
  pthread_mutex_unlock(&mutex1);
}</td></tr>
<tr><th colspan="4">Possible execution sequence</th></tr>
<tr><th>Thread 1</th><th>Thread 2</th><th>Thread 1</th><th>Thread 2</th></tr>
<tr><td>counter = 0</td><td>counter = 0</td><td>counter = 0</td><td>counter = 0</td></tr>
<tr><td>counter = 1</td><td>counter = 1</td><td>counter = 1</td><td>Thread 2 locked out.
Thread 1 has exclusive use of variable counter</td></tr>
<tr><td></td><td></td><td></td><td>counter = 2</td></tr>
</table>

รูป 2.6 เงื่อนไขแข่งขัน

2.6 ช่องทางข้อมูลอนุกรม (6)

2.6.1 บทนำช่องทางข้อมูลอนุกรม

ช่องทางข้อมูลอนุกรมจัดเป็นอุปกรณ์ไอ/โอ (I/O Device : Input/Output Device) ชนิดหนึ่งโดยอุปกรณ์ไอ/โอนี้เป็นหนทางที่จะรับข้อมูลเข้าคอมพิวเตอร์และส่งข้อมูลออกจากคอมพิวเตอร์ซึ่งอุปกรณ์ไอ/โอมีหลายชนิดเช่นช่องทางข้อมูลอนุกรม ช่องทางขนาน หน่วยขับแผ่นบันทึก แผงอีเทอร์เน็ต ยูเอสบี เป็นต้น เครื่องพีซีในปัจจุบันมักจะมีช่องทางข้อมูลอนุกรม 1 หรือ 2 ช่องโดยเป็นแบบ 9 พินหรืออาจเป็นแบบ 25 พินติดตั้งอยู่ด้านหลังของคอมพิวเตอร์โดยแต่ละช่องจะมีพินส่งข้อมูล 1 พินและพินรับข้อมูล 1 พินส่วนพินที่เหลือเป็นพินสำหรับสัญญาณควบคุมและพินสำหรับสายดินอ้างอิง

ช่องทางข้อมูลอนุกรมไม่ได้มีเพียงหัวต่อสำหรับเชื่อมต่อเท่านั้นแต่ยังต้องทำหน้าที่เปลี่ยนรูปแบบข้อมูลจากแบบขนานไปเป็นแบบอนุกรมและจากแบบอนุกรมกลับมาเป็นแบบขนานรวมถึงเปลี่ยนข้อมูลไปเป็นสัญญาณทางไฟฟ้าและเปลี่ยนสัญญาณทางไฟฟ้ากลับมาเป็นข้อมูลให้ได้ เพราะในเครื่องคอมพิวเตอร์การรับส่งบิตข้อมูลทำในรูปแบบขนานที่ใช้สายหลายสายในการรับหรือส่งพร้อม ๆ กันแต่การรับส่งแบบอนุกรมนั้นบิตข้อมูลจะส่งผ่านพินส่งที่ต่อกับสายส่งที่มีเพียง 1 เส้นและส่งได้ทีละ 1 บิตเท่านั้นส่วนการรับก็รับได้ทีละ 1 บิตผ่านพินรับที่ต่อกับสายรับที่มีเพียง 1 เส้นเช่นกันและด้วยเหตุที่กล่าวมาทำให้ช่องทางข้อมูลอนุกรมต้องสามารถเปลี่ยนข้อมูลระหว่างแบบขนานและแบบอนุกรมได้อย่างถูกต้องเหมาะสม

ชิ้นส่วนอิเล็คทรอนิกส์ส่วนใหญ่ที่พบในชิปของช่องทางข้อมูลอนุกรมนั้นรู้จักในชื่อ “ยูอาร์ต” (UART : Universal Asynchronous Receiver Transmitter)

2.6.2 การรับส่งไบต์ข้อมูล

2.6.2.1 การส่ง

คือการส่งข้อมูลออกจากช่องทางข้อมูลอนุกรมเพื่อส่งออกจากคอมพิวเตอร์ซึ่งงานส่วนใหญ่ทำโดยชิปที่ชื่อยูอาร์ต (UART : Universal Asynchronous Receiver Transmitter) โดยทำงานคู่กับโปรแกรมขับอุปกรณ์ (Device Driver) ที่ดำเนินงานอยู่บนซีพียูโดยการส่งไบต์ข้อมูลไปเลขที่อยู่ของช่องทางที่ต้องการส่งซึ่งไบต์ข้อมูลจะถูกส่งเข้าเรจิสเตอร์ขาส่ง(Transmit shift register) ทีละ 1 ไบต์และที่เรจิสเตอร์ขาส่งนี้เองจะส่งข้อมูลออกทีละ 1 บิตไปตามสายอนุกรมและเมื่อบิตสุดท้ายถูกส่งออกไปเรจิสเตอร์ขาส่งจะถามไปยังซีพียูว่ามีไบต์ต่อไปให้ส่งหรือไม่ถ้ามีซีพียูต้องส่งไบต์ถัดไปเข้าที่เรจิสเตอร์ขาส่งและทำเช่นนี้จนกว่าไม่มีไบต์ที่จะส่ง

แม้ว่าการส่งข้อมูลแบบที่กล่าวมาดูเหมือนจะไม่ซับซ้อนแต่ถ้าสังเกตให้ดีจะเห็นว่าในจังหวะที่เรจิสเตอร์ขาส่งถามหาไบต์ถัดไปกับซีพียูอาจเกิดเวลาหน่วงขึ้นถ้าซีพียูทำงานอื่นอยู่และไม่สามารถส่งไบต์ถัดไปให้เรจิสเตอร์ขาส่งได้ในทันทีที่ร้องขอและด้วยเหตุนี้จึงมีการพัฒนาเพื่อแก้เวลาหน่วงดังกล่าวโดยให้ช่องทางข้อมูลอนุกรมมีฮาร์ดแวร์บัฟเฟอร์ขาส่งเป็นของตัวเองเมื่อเรจิสเตอร์ขาส่งได้ส่งบิตสุดท้ายของไบต์ออกไปแล้วแทนที่จะไปถามหาไบต์ถัดไปกับซีพียูก็ไปถามกับฮาร์ดแวร์บัฟเฟอร์ขาส่งให้ส่งไบต์ถัดไปมาแทนเพราะฮาร์ดแวร์บัฟเฟอร์ขาส่งมีอยู่ในตัวช่องทางข้อมูลอนุกรมเองโอกาสเกิดเวลาหน่วงจึงลดลงมาก

แรกเริ่มเดิมทีฮาร์ดแวร์บัฟเฟอร์ขาส่งมีขนาด 1 ไบต์ซึ่งมีลักษณะการทำงานคือเมื่อฮาร์ดแวร์บัฟเฟอร์ขาส่งส่งไบต์ให้เรจิสเตอร์ขาส่งแล้วและต้องการไบต์ถัดไปมาเก็บไว้ ฮาร์ดแวร์บัฟเฟอร์ขาส่งก็จะส่งสัญญาณขัดจังหวะไปที่ซีพียูด้วยการใส่แรงดันไฟฟ้าที่เหมาะสมบนสายไฟเส้นที่กำหนดไว้บนบัสคอมพิวเตอร์เพื่อให้ซีพียูส่งไบต์ถัดไปมาที่ฮาร์ดแวร์บัฟเฟอร์ขาส่ง

เมื่อซีพียูได้รับสัญญาณขัดจังหวะแล้วซีพียูจะตรวจว่าอุปกรณ์ตัวใด (ในกรณีไม่ใช้สัญญาณขัดจังหวะร่วม) หรืออุปกรณ์กลุ่มใด (ในกรณีใช้สัญญาณขัดจังหวะร่วม) ร้องขอการบริการจากซีพียูซึ่งในกรณีนี้ก็คืออุปกรณ์ช่องทางข้อมูลอนุกรมที่ร้องขอการบริการ ซีพียูจะดำเนินงานโปรแกรมขับอุปกรณ์ของช่องทางข้อมูลอนุกรมให้ตรวจสอบเรจิสเตอร์เลขที่อยู่ช่องทางนั้น ๆ เพื่อดูว่ามีอะไรเกิดขึ้นซึ่งในกรณีนี้โปรแกรมขับอุปกรณ์จะพบว่าฮาร์ดแวร์บัฟเฟอร์ขาส่งมีที่ว่างสำหรับไบต์ถัดไปอยู่และหากโปรแกรมขับอุปกรณ์ยังมีไบต์ข้อมูลที่ต้องการส่งอยู่อีกโปรแกรมขับอุปกรณ์ก็จะส่งไบต์ข้อมูลถัดไปให้ฮาร์ดแวร์บัฟเฟอร์ขาส่งต่อไป

สรุปแล้วถ้าไบต์ข้อมูลในเรจิสเตอร์ขาส่งถูกส่งออกไปจนหมดจะเกิดเหตุการณ์ 3 เหตุการณ์ดังนี้

1. ไบต์ข้อมูลจะเคลื่อนจากฮาร์ดแวร์บัฟเฟอร์ขาส่งไปที่เรจิสเตอร์ขาส่ง

2. เรจิสเตอร์ขาส่งส่งข้อมูลแบบบิตต่อบิตออกไปตามสายส่ง

3. ฮาร์ดแวร์บัฟเฟอร์ขาส่งส่งสัญญาณขัดจังหวะบอกให้โปรแกรมขับอุปกรณ์ส่งไบต์ข้อมูลถัดไปมาที่ฮาร์ดแวร์บัฟเฟอร์ขาส่ง

แต่การทำเช่นนี้จะเห็นว่าฮาร์ดแวร์บัฟเฟอร์ขาส่งจะต้องส่งสัญญาณขัดจังหวะให้ซีพียูทุก ๆ การส่งข้อมูล 1 ไบต์และโดยทั่วไปในแต่ละไบต์ที่ส่งมักจะมีบิตเริ่ม (start bit) บิตภาวะคู่หรือคี่ (parity bit) บิตหยุด (stop bit) เพิ่มเข้ามาทำให้เฉลี่ยแล้ว 1 ไบต์จะมีประมาณ 10 บิตและเมื่อลองคำนวณที่อัตราส่ง 9,600 บิต/วินาทีหรือคิดเป็น 960 ไบต์ต่อวินาทีจะเห็นได้ว่าในกรณีที่มีข้อมูลส่งออกช่องทางข้อมูลอนุกรมนี้ตลอดเวลาซีพียูจะต้องรับสัญญาณขัดจังหวะจากงานส่วนนี้เพียงส่วนเดียวสูงถึง 960 ครั้งต่อวินาที

เพื่อลดจำนวนสัญญาณขัดจังหวะที่ส่งถึงซีพียูในปัจจุบันช่องทางข้อมูลอนุกรมส่วนใหญ่จึงเพิ่มขนาดฮาร์ดแวร์บัฟเฟอร์ขาส่งจาก 1 ไบต์เป็น 16 ไบต์หรืออาจเป็นขนาดอื่นขึ้นกับฮาร์ดแวร์ ซึ่งการเพิ่มขนาดของฮาร์ดแวร์บัฟเฟอร์ขาส่งนี้ทำให้การส่งสัญญาณขัดจังหวะถึงซีพียูจะมีความถี่ลดลงโดยในกรณีความจุของฮาร์ดแวร์บัฟเฟอร์ขาส่ง 16 ไบต์การส่งสัญญาณขัดจังหวะอาจส่งเมื่อไบต์ที่ 16 ถูกส่งเข้าเรจิสเตอร์ขาส่งแล้วฮาร์ดแวร์บัฟเฟอร์ขาส่งค่อยส่งสัญญาณขัดจังหวะให้ซีพียูเพื่อขอไบต์ข้อมูลชุดถัดไปซึ่งอาจมา 16 ไบต์ต่อการสัญญาณขัดจังหวะ 1 ครั้งและจากการทำงานของฮาร์ดแวร์บัฟเฟอร์ที่มีลักษณะไบต์ไหนที่เข้ามาก่อนจะถูกส่งออกก่อนจึงมักเรียกฮาร์ดแวร์บัฟเฟอร์ว่า “ไฟโฟ” (FIFO : first-in-first-out)

ถึงแม้ว่าจะมีการลดจำนวนสัญญาณขัดจังหวะให้น้อยลงตามขนาดของไฟโฟแต่การส่งไบต์ข้อมูลจากโปรแกรมขับอุปกรณ์มายังไฟโฟผ่านบัสข้อมูลยังคงเป็นแบบครั้งละ 1 ไบต์ซึ่งเมื่อเทียบความกว้างบัสข้อมูลที่มีขนาด 16 บิต 32 บิต 64 บิตตามแต่สถาปัตยกรรมกับข้อมูล 1 ไบต์ที่มีเพียง 8 บิตแล้วอาจมองได้ว่าเป็นการใช้งานบัสข้อมูลที่ไม่คุ้มค่า

2.6.2.2 การรับ

มีลักษณะเหมือนกับการส่งแตกต่างเพียงทิศทางของไบต์ข้อมูลที่ตรงข้ามกับการส่งเท่านั้น โดยเริ่มจากเรจิสเตอร์ขารับรับข้อมูลครบ 1 ไบต์แล้วจึงย้ายข้อมูลไปสู่ไฟโฟขารับที่มีความจุ 1 ไบต์ หลังจากนั้นไฟโฟขารับจะส่งสัญญาณขัดจังหวะให้ซีพียูและเมื่อซีพียูได้รับสัญญาณขัดจังหวะแล้วซีพียูจะดำเนินงานโปรแกรมขับอุปกรณ์ของช่องทางข้อมูลอนุกรมให้ตรวจสอบเรจิสเตอร์เลขที่อยู่ช่องทางนั้น ๆ เพื่อดูว่ามีอะไรเกิดขึ้นซึ่งในกรณีนี้โปรแกรมขับอุปกรณ์จะพบว่าไฟโฟขารับมีไบต์ข้อมูลอยู่และจะต้องย้ายไบต์ข้อมูลออกมาเพื่อให้มีที่ว่างสำหรับไบต์ข้อมูลถัดไปที่เรจิสเตอร์ขารับจะส่งมาเก็บ

สำหรับช่องทางข้อมูลอนุกรมที่มีไฟโฟขารับขนาด 16 ไบต์การส่งสัญญาณขัดจังหวะอาจทำเมื่อมีไบต์ข้อมูลในไฟโฟขารับ 14 ไบต์โดยในช่วงที่รอโปรแกรมขับอุปกรณ์มาย้ายข้อมูลออกไปนั้นเรจิสเตอร์ขารับอาจส่งไบต์ถัดไปมาเพิ่มได้ไม่เกิน 2 ไบต์ก่อนที่ข้อมูลจะถูกย้ายออกไปหากไม่เช่นนั้นไฟโฟขารับจะอยู่ในสถานะเกินการดำเนินงาน (Overrun) ทำให้ข้อมูลเกิดความผิดพลาดขึ้นได้

2.6.2.3 บัฟเฟอร์ขนาดใหญ่ของช่องทางข้อมูลอนุกรม

เป็นบัฟเฟอร์ที่อยู่ในหน่วยความจำหลัก (Main Memery) อาจมีขนาดถึง 8,000 ไบต์โดยทำหน้าที่เป็นจุดพักข้อมูลขารับและขาส่งโดยข้อมูลขารับที่โปรแกรมขับอุปกรณ์ย้ายจากไฟโฟขา

รับจะมาพักที่จุดนี้ก่อนเพื่อรอที่จะ read() ออกไป ส่วนข้อมูลขาส่งที่มาจาก write() ก็จะมาพักที่จุดนี้เช่นกันก่อนที่โปรแกรมขับอุปกรณ์จะมาย้ายไปสู่ไฟโฟขาส่ง

2.6.3 **ความรู้พื้นฐานของช่องทางข้อมูลอนุกรม**

ในส่วนนี้จะกล่าวถึงพื้นฐานต่าง ๆ ที่เกี่ยวข้องกับช่องทางข้อมูลอนุกรมถึงแม้ว่าการใช้งานช่องทางข้อมูลอนุกรมนั้นสามารถทำได้โดยไม่ต้องอาศัยความรู้ในส่วนที่จะกล่าวต่อไปนี้เลยก็ตามแต่ความเข้าใจพื้นฐานที่ดีพอก็จะมีส่วนช่วยให้การหาสาเหตุความผิดพลาดตลอดถึงวิธีแก้ปัญหาในระหว่างการพัฒนาโปรแกรมเพื่อนำไปใช้งานได้อย่างถูกต้องและรวดเร็วโดยเนื้อหาในส่วนนี้อาจมีบางส่วนซ้ำกับที่กล่าวมาแล้วแต่จะมีการเพิ่มเติมรายละเอียดให้สมบูรณ์มากขึ้น

2.6.3.1 **พินและสาย**

ในอดีตช่องทางข้อมูลอนุกรมมักเป็นแบบ 25 พินแต่ปัจจุบันจะเหลือเป็นแบบ 9 พินมากกว่าซึ่งพินแต่พินจะมีการเชื่อมต่อกับสายโดยแยกเป็น 1 พินส่ง 1 พินรับและที่เหลือเป็นพินสัญญาณควบคุมกับพินสำหรับสายดินอ้างอิงซึ่งสัญญาณทุกสัญญาณต้องวัดเทียบกับสายดินอ้างอิงนี้เพื่อแยกว่าเป็นสถานะออนหรือออฟส่วนจำนวนพินหรือสายที่น้อยที่สุดที่ใช้ในการสื่อสารทั้งรับและส่งคือ 3 พินหรือ 3 สายโดยมีสายรับ สายส่ง สายดินอ้างอิงหรือในบางครั้งอาจรับส่งได้โดยไม่ใช้สายดินอ้างอิงเลยก็ได้ซึ่งกรณีนี้อาจมีความผิดพลาดในการรับส่งมากขึ้น ส่วนสายที่เหลือนอกเหนือ 3 สายที่กล่าวก็เป็นสายสัญญาณที่ใช้ควบคุมการรับส่งให้เป็นไปอย่างถูกต้องและถึงแม้ว่าสัญญาณควบคุมเหล่านี้สามารถรับส่งร่วมกันได้บนสายเพียงเส้นเดียวแต่ช่องทางข้อมูลอนุกรมก็ยังแยกสายให้แต่ละสัญญาณควบคุมมีสายรับส่งสัญญาณเป็นของตัวเองโดยสายสัญญาณควบคุมนี้มักเรียกว่า “เส้นควบคุมโมเด็ม” (Modem Control Lines) และสายควบคุมโมเด็มนี้จะมีได้ 2 สถานะคือแทนสถานะออนด้วยแรงดันไฟฟ้า +12 โวลต์และแทนสถานะออฟด้วยแรงดันไฟฟ้า -12 โวลต์ซึ่งสายควบคุมโมเด็มนี้จะมีอยู่สายหนึ่งที่สามารถบอกให้คอมพิวเตอร์หยุดส่งข้อมูลออกได้และจะมีอีกสายหนึ่งที่สามารถบอกให้อุปกรณ์ที่ต่ออยู่กับคอมพิวเตอร์หยุดส่งข้อมูลมาได้เช่นกัน

2.6.3.2 **อาร์เอส-232 หรืออีไอเอ-232**

ช่องทางข้อมูลอนุกรม (ไม่รวมยูเอสบี) หรืออาร์เอส-232-ซี (RS-232-C) อีไอเอ-232-ดี (EIA-232-D) อีไอเอ-232-อี (EIA-232-E) ทั้งสามชื่อนี้เกือบจะเป็นชื่อเรียกสิ่งเดียวกันโดยคำนำหน้า “อาร์เอส” (RS : Recommended Standard) นั้นมาจากอีไอเอ (EIA : Electronics

Industries Association) และเปลี่ยนมาเป็นอีไอเอ/ทีไอเอ (EIA/TIA) หลังจากอีไอเอรวมกับทีไอเอ (TIA : Telecommunications Industries Association)

2.6.3.3 เลขที่อยู่ไอ/โอและสัญญาณขัดจังหวะ

เมื่อคอมพิวเตอร์ต้องการติดต่อกับช่องทางข้อมูลอนุกรมระบบปฏิบัติการจะต้องทราบว่ามีช่องทางข้อมูลอนุกรมต่ออยู่ในคอมพิวเตอร์หรือไม่และถ้ามีแล้วจะต้องทราบว่าช่องทางนั้นอยู่ที่ไหนซึ่งในที่นี้หมายถึงเลขที่อยู่ไอ/โอ (I/O Address) ของช่องทางนั้นเองรวมถึงจะต้องทราบสายสัญญาณที่ช่องทางข้อมูลอนุกรมใช้ส่งสัญญาณขัดจังหวะเพื่อร้องขอการบริการจากซีพียูหรือที่มักเรียกว่า “เลขไออาร์คิว” (IRQ Number) ทั้งเลขที่อยู่ไอ/โอและเลขไออาร์คิวของแต่ละช่องทางข้อมูลอนุกรมจะถูกเก็บไว้ที่หน่วยความจำถาวร (Non-Volatile Memory) โดยทั่วไปบัสพีซีไอ (PCI Bus) จะมีระบบขัดจังหวะเป็นของตัวเองโดยมีจุดเด่นที่อนุญาตให้มีอุปกรณ์มากกว่า 1 ชิ้นสามารถใช้เลขสัญญาณขัดจังหวะร่วมกันได้

ส่วนการส่งข้อมูลจากหน่วยความจำหลักไปสู่อุปกรณ์หรือไปสู่ช่องทางสามารถทำได้โดยการใส่เลขที่อยู่ไอ/โอในบัสเลขที่อยู่ของคอมพิวเตอร์ทำให้อุปกรณ์หรือช่องทางที่มีเลขที่อยู่ไอ/โอต้องตรวจสอบว่าเลขที่อยู่ไอ/โอในบัสเลขที่อยู่นั้นตรงกับเลขที่อยู่ไอ/โอของตัวเองหรือไม่ ถ้าเลขที่อยู่ไอ/โอตรงกันแสดงว่าหน่วยความจำหลักส่งข้อมูลบางอย่างมาให้อุปกรณ์หรือช่องทางนั้น ๆ ซึ่งอุปกรณ์หรือช่องทางดังกล่าวจะต้องทำการอ่านข้อมูลออกจากบัสข้อมูลทันที

เลขที่อยู่ไอ/โอของแต่ละอุปกรณ์หรือแต่ละช่องทางนั้นจะมีเลขที่อยู่เป็นช่วงโดยค่าเลขที่อยู่ที่มีค่าต่ำที่สุดของช่วงจะเรียกว่า “เลขที่อยู่ฐาน” (Base Address) และในที่นี้การกล่าวหรืออ้างถึงคำว่า “เลขที่อยู่” นั้นจะหมายถึง “เลขที่อยู่ฐาน” เสมอ

2.6.3.4 ชื่อช่องทาง ttyS0 ttyS1 ฯลฯ

สำหรับช่องทางข้อมูลอนุกรมแต่ละช่องทางจะใช้ชื่อ ttyS0 ttyS1 ฯลฯ (เทียบได้กับ COM1 COM2 ฯลฯ ในดอสหรือวินโดวส์) ซึ่งชื่อนี้จัดเป็นแฟ้มพิเศษใน /dev สามารถเรียกดูได้โดย “ls -la /dev/ttyS*” และชื่อแต่ละชื่อนี้จะใช้แทนหรือเทียบได้กับช่องทางข้อมูลอนุกรมที่ต่ออยู่กับคอมพิวเตอร์โดยโปรแกรมขับอุปกรณ์ของช่องทางข้อมูลอนุกรมจะมีตารางดูแลชื่อช่องทางว่าตรงกับเลขที่อยู่ไอ/โอและเลขไออาร์คิวไหนซึ่งค่าเลขที่อยู่ไอ/โอกับเลขไออาร์คิวนี้สามารถตั้งค่าได้โดยคำสั่ง “setserial” แต่คำสั่งนี้สามารถตั้งค่าเลขที่อยู่ไอ/โอกับเลขไออาร์คิวทางซอฟต์แวร์เท่านั้นซึ่งในบางกรณีมีความจำเป็นต้องมีการตั้งค่าทางฮาร์ดแวร์ด้วยโดยการเซตตัวโยงหรือดิปสวิตช์ซึ่งการตั้งค่าทั้งทางซอฟต์แวร์และฮาร์ดแวร์นั้นจำเป็นจะต้องตรงกันเพื่อการทำงานที่ถูกต้อง

2.6.3.5 การขัดจังหวะ

เมื่อช่องทางข้อมูลอนุกรมได้รับไบต์ข้อมูลเข้ามาเก็บในไฟโฟขารับตามจำนวนไบต์ที่ตั้งไว้แล้วซึ่งอาจจะเป็น 1 ไบต์ 4 ไบต์ 8 ไบต์หรือ 14 ไบต์จากนั้นไฟโฟขารับจะส่งสัญญาณให้ซีพียูโดยการส่งสัญญาณไฟฟ้าที่เรียกว่า “สัญญาณขัดจังหวะ” ไปบนสายสัญญาณที่แน่นอนเส้นหนึ่งที่จะส่งสัญญาณขัดจังหวะดังกล่าวเข้าสู่ซีพียูต่อไป แต่อย่างไรก็ตามการส่งสัญญาณขัดจังหวะอาจเกิดขึ้นได้แม้ว่าจำนวนไบต์ในไฟโฟขารับจะยังไม่ถึงจำนวนที่ตั้งไว้แต่ระยะเวลาที่รอไบต์ถัดไปนั้นนานเกินระยะเวลาที่กำหนดหรือ “หมดเวลารอ” (timeout) ดังนั้นมีความเป็นไปได้ที่การส่งสัญญาณขัดจังหวะอาจเกิดขึ้นทุก ๆ ไบต์หากไบต์ข้อมูลที่ได้รับมีระยะเวลาห่างกันมากกว่าระยะหมดเวลารอที่กำหนด (เช่นไบต์ข้อมูลที่มาจากแป้นพิมพ์ที่มักต้องรอการกดแป้นพิมพ์จากผู้พิมพ์) สำหรับชิปยูอาร์ตก็มีการกำหนดระยะหมดเวลารอเช่นกันตัวอย่างเช่นเมื่อได้รับไบต์ข้อมูลเข้ามา 4 ไบต์ภายในระยะเวลาที่กำหนดจากนั้นจะรอไบต์ถัดไปจนหมดเวลารอชิปยูอาร์ตก็จะส่งสัญญาณขัดจังหวะให้ซีพียูมาดึงไบต์ข้อมูล 4 ไบต์นั้นออกไปจากไฟโฟขารับทันทีโดยไม่ต้องรอให้ครบจำนวนไบต์ตามที่ตั้งไว้ แต่ในกรณีที่ไม่มีไบต์ข้อมูลในไฟโฟขารับเลยก็จะไม่มีการส่งสัญญาณขัดจังหวะให้ซีพียูแม้จะหมดเวลารอแล้วก็ตาม

สายนำสัญญาณที่นำสัญญาณขัดจังหวะระหว่างอุปกรณ์หรือช่องทางกับซีพียูนั้นจะต้องมีหมายเลขกำกับที่เรียกว่า “ไออาร์คิว” (IRQ : Interrupt ReQuest) และดังที่กล่าวมาแล้วช่องทางข้อมูลอนุกรมจะต้องทราบสายนำสัญญาณหรือไออาร์คิวของตัวเองเพื่อใช้รับส่งสัญญาณขัดจังหวะตัวอย่าง ttyS0 ปกติจะใช้ไออาร์คิวหมายเลข 4 หรือไออาร์คิว 4 (IRQ4) และช่องทางข้อมูลอนุกรมจะส่งสัญญาณขัดจังหวะไปหาซีพียูก็เพื่อร้องขอการบริการจากซีพียูโดยเฉพาะการรับไบต์ข้อมูลที่ไฟโฟขารับในช่องทางข้อมูลอนุกรมนั้นมีพื้นที่รองรับไบต์ได้เพียง 16 ไบต์หากซีพียูมาย้ายไบต์ข้อมูลออกจากไฟโฟขารับไม่ทันก็จะทำให้ไม่มีที่ว่างสำหรับไบต์ข้อมูลถัดไปที่จะรับเข้ามาซึ่งในกรณีนี้จะเกิด “การล้น” (overflow) หรือ “เกินการดำเนินงาน” (overrun) ทำให้ไบต์ข้อมูลส่วนนั้น ๆ สูญหายไปซึ่ง ณ ตอนนี้ยังไม่มีการควบคุมสายงานใดที่สามารถป้องกันเหตุการณ์นี้ได้

การส่งสัญญาณขัดจังหวะนอกจากจะใช้เมื่อมีการรับข้อมูลเข้ามาแล้วการส่งข้อมูลออกก็ต้องใช้สัญญาณขัดจังหวะเช่นเดียวกันโดยเมื่อไฟโฟขาส่งที่ใช้พักไบต์ข้อมูลที่ต้องการส่งได้ทำการส่งไบต์ข้อมูลให้เรจิสเตอร์ขาส่งจนหมดแล้วจะทำมีที่ว่างในไฟโฟขาส่งซึ่งในที่นี้จะใช้ค่า 16 ไบต์จากนั้นไฟโฟขาส่งจะส่งสัญญาณขัดจังหวะให้ซีพียูทราบว่ามีที่ว่างในไฟโฟขาส่งและเมื่อซีพียูทราบและมีข้อมูลที่ต้องการส่งอีกซีพียูจะส่งข้อมูล 16 ไบต์ถัดไปหรืออาจน้อยกว่าในกรณีที่ไบต์ข้อมูลที่เหลือมีไม่ถึง 16 ไบต์ไปให้กับไฟโฟขาส่งเพื่อทำการส่งให้เรจิสเตอร์ขาส่งต่อไป

ที่กล่าวมาข้างต้นว่าช่องทางข้อมูลอนุกรมส่งสัญญาณขัดจังหวะให้ซีพียูนั้นแท้จริงแล้วสัญญาณขัดจังหวะที่ออกจากช่องทางข้อมูลอนุกรมถูกส่งไปที่ชิปที่เรียกว่า “ตัวควบคุมการขัดจังหวะ” (Interrupt Controller) เพื่อแจ้งว่าช่องทางข้อมูลอนุกรมนั้นต้องการใช้บริการจากซีพียู หลังจากนั้นตัวควบคุมการขัดจังหวะค่อยส่งสัญญาณให้ซีพียูอีกทอดหนึ่งและเมื่อซีพียูได้รับสัญญาณแล้วก็จะดำเนินงานโปรแกรมพิเศษเพื่อให้บริการกับช่องทางข้อมูลอนุกรมดังกล่าวโดยเรียกโปรแกรมนั้นว่า “รูทีนบริการสัญญาณขัดจังหวะ” (Interrupt Service Routine) ซึ่งเป็นส่วนหนึ่งของโปรแกรมขับช่องทางข้อมูลอนุกรมโดยโปรแกรมนี้จะตรวจว่าเกิดอะไรขึ้นที่ช่องทางข้อมูลอนุกรมดังกล่าวแล้วจึงเข้าให้บริการอาทิเช่นช่องทางข้อมูลอนุกรมนั้นต้องการให้ย้ายข้อมูลออกจากหรือเข้าไปสู่ไฟโฟเป็นต้นส่วนการตรวจว่ามีอะไรเกิดขึ้นที่ช่องทางข้อมูลอนุกรมตัวรูทีนบริการสัญญาณขัดจังหวะสามารถตรวจได้โดยง่ายเพราะช่องทางข้อมูลอนุกรมทุกช่องทางจะมีเรจิสเตอร์เลขที่อยู่ไอ/โอของแต่ละช่องทางซึ่งโปรแกรมขับช่องทางข้อมูลอนุกรมนั้นทราบค่าเลขที่อยู่ไอ/โอดังกล่าวอยู่แล้วโดยเรจิสเตอร์ดังกล่าวจะมีข้อมูลสถานะของช่องทางข้อมูลอนุกรมที่ต่ออยู่ ดังนั้นรูทีนบริการสัญญาณขัดจังหวะเพียงอ่านข้อมูลจากเรจิสเตอร์ก็จะทราบว่าเกิดอะไรขึ้นที่ช่องทางข้อมูลอนุกรมเพื่อจะได้ให้บริการที่เหมาะสมต่อไป

2.6.3.6 **กระแสข้อมูล**

คือข้อมูลที่ผ่านเข้าหรือออกจากช่องทางซึ่งเรียกปริมาณหรือจำนวนข้อมูลเข้าหรือออกนี้ว่า “อัตราสายงาน” (Flow rate) มีหน่วยเป็นบิต/วินาที (bits per second) บางครั้งมักเรียกปริมาณข้อมูลเข้าหรือออกนี้ผิดไปเป็น “ความเร็ว” (Speed) แต่ส่วนใหญ่เมื่อกล่าวถึงคำว่าความเร็วก็เป็นที่เข้าใจกันว่าคืออัตราสายงานนั้นเอง

สิ่งสำคัญที่ต้องเข้าใจอีกประการคือความเร็วเฉลี่ยมักจะมีค่าน้อยกว่าความเร็วที่กำหนดเสมอเพราะความเร็วที่กำหนดเช่น 9,600 บิต/วินาทีนั้นคือความเร็วสูงสุดที่ช่องทางสามารถรับหรือส่งได้ถ้าต้องการให้ความเร็วเฉลี่ยมีค่าเท่ากับความเร็วที่กำหนดการรับส่งต้องทำอยู่ตลอดเวลา

2.6.3.7 **การควบคุมสายงาน**

เป็นการควบคุมจำนวนไบต์ข้อมูลที่จะรับส่งผ่านช่องทางให้อยู่ในระดับที่เหมาะสม ซึ่งการควบคุมช่องทางข้อมูลอนุกรมนั้นทำโดยการสั่งให้ช่องทางหยุดหรือสั่งให้ช่องทางส่งไบต์ข้อมูลซึ่งการทำเช่นนี้จะไม่ทำให้ข้อมูลที่รับส่งมีการสูญหายแต่อย่างใดและการควบคุมสายงานก็มีความจำเป็นต้องใช้เมื่อการรับส่งนั้นเกิดระหว่างคอมพิวเตอร์กับโมเด็มหรือฮาร์ดแวร์บางชนิด

2.6.3.7.1 ตัวอย่างการควบคุมสายงาน

พิจารณาการต่อช่องทางข้อมูลอนุกรมผ่านสายเคเบิลสั้นเข้ากับโมเด็ม (แบบภายนอก) และใช้โมเด็มนี้ต่อเข้ากับสายโทรศัพท์ที่ความเร็ว 33,600 บิต/วินาทีโดยไม่มีกระบวนการบีบอัดและแก้ไขข้อมูลผิดพลาดแต่อย่างใดและขณะเดียวกันช่องทางข้อมูลอนุกรมที่เครื่องคอมพิวเตอร์ถูกตั้งความเร็วรับส่งที่ 115,200 บิต/วินาทีเพื่อทำการส่งข้อมูลจากคอมพิวเตอร์ไปยังสายโทรศัพท์ ซึ่งในที่นี้จะเห็นได้ว่าความเร็วข้อมูลในสายเคเบิลสั้นที่เข้าสู่โมเด็มจะอยู่ที่ 115,200 บิต/วินาทีส่วนความเร็วข้อมูลออกไปสู่สายโทรศัพท์จะอยู่ที่ 33,600 บิต/วินาทีหรือคิดเป็นส่วนต่าง 115,200-33,600=81,600 บิต/วินาทีซึ่งทำให้บัฟเฟอร์อยู่ในสถานะเกินการดำเนินงาน

แต่ในกรณีที่มีการควบคุมสายงาน เมื่อบัฟเฟอร์ในโมเด็มมีไบต์ข้อมูลเกือบเต็มความจุแล้วโมเด็มจะส่งสัญญาณ “หยุด” (stop) ให้ช่องทางข้อมูลอนุกรมเมื่อช่องทางข้อมูลอนุกรมได้รับสัญญาณหยุดแล้วก็จะทำการส่งสัญญาณนั้นต่อให้โปรแกรมขับอุปกรณ์เพื่อให้ “ชะงัก” (Halt) การส่งข้อมูลทันทีขณะเดียวกันนั้นโมเด็มก็จะทยอยส่งข้อมูลในบัฟเฟอร์ออกไปทางสายโทรศัพท์ที่ความเร็ว 33,600 บิต/วินาทีจนไบต์ข้อมูลในบัฟเฟอร์ใกล้หมดโมเด็มจะส่งสัญญาณ “เริ่ม” (Start) ให้ช่องทางข้อมูลอนุกรมเพื่อผ่านสัญญาณไปสู่โปรแกรมขับอุปกรณ์ให้เริ่มส่งข้อมูลที่ 115,200 บิต/วินาทีต่อไปซึ่งเรียกการควบคุมสายงานแบบนี้ว่า “การควบคุมสายงานแบบเริ่ม/หยุด” (Start Stop Flow Control)

ในกรณีตัวอย่างข้างบนนั้นโมเด็มไม่มีการบีบอัดข้อมูลซึ่งอาจเป็นเพราะข้อมูลที่รับมานั้นถูกบีบอัดแล้วและไม่สามารถบีบอัดเพิ่มได้อีกแต่ถ้าข้อมูลนั้นสามารถบีบอัดได้และโมเด็มมีความสามารถในการบีบอัดสูงเช่นในกรณีข้างบนที่ความเร็วข้อมูลเข้าและออกจากโมเด็มมีค่า 115,200 และ 33,600 บิต/วินาทีตามลำดับโมเด็มจะต้องมีความสามารถในการบีบอัดข้อมูล 115,200/33,600 หรือประมาณ 3.43 ซึ่งในกรณีที่โมเด็มทำได้ก็ไม่มีความจำเป็นที่ต้องใช้การควบคุมสายงานแต่การบีบอัดนั้นปกติจะมีอัตราการบีบอัดที่ไม่แน่นอนสามารถเปลี่ยนแปลงได้ตลอดเวลาและถ้าอัตราบีบอัดต่ำกว่า 3.43 เมื่อไหร่ระบบก็มีความจำเป็นต้องใช้การควบคุมสายงานในช่วงเวลาดังกล่าว

ในตัวอย่างที่ผ่านมาโมเด็มที่ใช้อธิบายเป็นแบบภายนอกแต่เหตุการณ์ที่กล่าวมาก็ยังคงเกิดกับโมเด็มแบบภายในคือข้อจำกัดในเรื่องความเร็วแม้ว่าโมเด็มแบบภายในจะไม่ต้องใช้สายเคเบิลสั้นเหมือนโมเด็มภายนอกก็ตามที

การควบคุมสายงานนั้นไม่ได้ใช้เฉพาะเวลาส่งข้อมูลจากคอมพิวเตอร์ไปสู่โมเด็มเท่านั้นในทิศทางตรงข้ามหรือเวลาโมเด็มส่งข้อมูลเข้าคอมพิวเตอร์ก็มีการควบคุมสายงานเช่นเดียวกันซึ่งแต่ละทิศทางการส่งจะมีบัฟเฟอร์ที่เกี่ยวข้องอยู่ 3 บัฟเฟอร์คือบัฟเฟอร์ในโมเด็ม ไฟโฟในยูอาร์ตบัฟเฟอร์ในหน่วยความจำหลักที่ดูแลจัดการโดยโปรแกรมขับช่องทางข้อมูลอนุกรมโดยการควบคุม

สายงานจะทำหน้าที่ป้องกันการล้นของบัฟเฟอร์ภายในโมเด็ม และในหน่วยความจำหลักเท่านั้นไม่สามารถป้องกันการล้นที่จะเกิดกับไฟโฟในยูอาร์ตได้

ในลินุกซ์ไฟโฟในยูอาร์ตนั้นจะไม่ได้รับการปกป้องจากการควบคุมสายงานแต่จะอาศัยกระบวนการขัดจังหวะที่น่าเชื่อถือได้ว่าจะไม่เกิดการล้นในส่วนนี้และถึงแม้ว่าชิปยูอาร์ตบางตัวถูกออกแบบและสร้างให้มีการควบคุมสายงานที่สามารถป้องกันการล้นในตัวเองได้แต่ในขณะนี้ลินุกซ์ดูเหมือนว่าจะยังไม่รองรับกับระบบดังกล่าว

2.6.3.7.2 อาการที่เกิดเมื่อไม่มีการควบคุมสายงาน

ในกรณีที่ความเร็วการรับส่งข้อมูลไม่เหมาะสมหรือข้อมูลเข้ามากกว่าข้อมูลออกและไม่มีการควบคุมสายงานจะทำให้ข้อมูลบางส่วนเกิดการสูญหาย

2.6.3.7.3 การควบคุมสายงานด้วยฮาร์ดแวร์กับการควบคุมสายงานด้วยซอฟต์แวร์

การควบคุมสายงานด้วยฮาร์ดแวร์จะทำโดยการส่งสัญญาณผ่านสายควบคุมโมเด็ม (Modem Control Wires) โดยสัญญาณที่ส่งนั้นมี 2 คำสั่งคือ “เริ่ม” กับ “หยุด” ผ่านพิน 2 พิน คือพินอาร์ทีเอส (RTS : Request To Send) กับพินซีทีเอส (CTS : Clear To Send) เมื่อคอมพิวเตอร์พร้อมที่จะรับข้อมูลคอมพิวเตอร์จะใส่แรงดันไฟฟ้าบวก (+12 โวลต์หรืออาจเป็นค่าอื่นขึ้นกับมาตรฐานที่ใช้) ที่พินอาร์ทีเอสเพื่อบอกว่า “ขอร้องให้ส่งข้อมูลมา” (Request To Send to me) แต่ถ้าคอมพิวเตอร์ไม่พร้อมหรือไม่สามารถรับไบต์ข้อมูลได้คอมพิวเตอร์จะใส่แรงดันลบ (-12 โวลต์หรืออาจเป็นค่าอื่นขึ้นกับมาตรฐานที่ใช้) ที่พินอาร์ทีเอสเพื่อบอกว่า “ให้หยุดการส่งข้อมูลมา” (Stop Sending to me) โดยพินอาร์ทีเอสนี้จะต่อกับสายเคเบิลไปสู่พินของโมเด็มหรือพินของอุปกรณ์อื่นโดยพินที่จะต่อด้วยนั้นต้องทำหน้าที่คอยรับสัญญาณเพียงอย่างเดียวเท่านั้น

ในกรณีต่อระหว่างคอมพิวเตอร์กับโมเด็ม พินอาร์ทีเอสจากคอมพิวเตอร์จะต่อผ่านสายเคเบิลไปสู่พินอาร์ทีเอสของโมเด็มแต่หากเป็นการต่อระหว่างคอมพิวเตอร์กับเครื่องพิมพ์หรือต่อกับเครื่องคอมพิวเตอร์เครื่องอื่นหรือต่อกับอุปกรณ์ที่ไม่ใช่โมเด็มพินอาร์ทีเอสจากคอมพิวเตอร์จะต้องต่อผ่านสายเคเบิลไปสู่พินซีทีเอสของอุปกรณ์เหล่านั้นโดยการต่อแบบหลังนี้จะต้องการสายแบบ “ครอสโอเวอร์” (crossover) หรือใช้อุปกรณ์ “โมเด็มว่าง” (null modem) ซึ่งสายแบบนี้จะสลับพินระหว่างอาร์ทีเอสกับซีทีเอสให้ซึ่งแตกต่างกับการต่อระหว่างคอมพิวเตอร์กับโมเด็มที่สายต่อไม่ต้องสลับระหว่างสองพินดังกล่าวหรือที่เรียกว่าการต่อตรงนั้นเอง

ในทิศทางตรงข้ามโมเด็มจะใช้การควบคุมสายงานด้วยฮาร์ดแวร์ผ่านพินซีทีเอสของโมเด็มไปสู่พินซีทีเอสของคอมพิวเตอร์และสำหรับอุปกรณ์ที่ไม่ใช่โมเด็มจะใช้การควบคุมสายงานด้วยฮาร์ดแวร์ผ่านพินอาร์ทีเอสต่อผ่านสายเคเบิลไปสู่พินซีทีเอสของคอมพิวเตอร์ แต่อย่างไรก็ตามการ

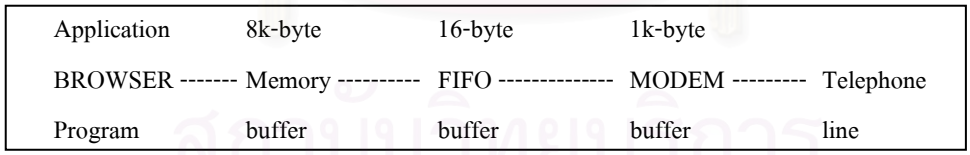
ควบคุมสายงานด้วยฮาร์ดแวร์อาจจะใช้พินอื่นในการรับส่งสัญญาณก็ได้เช่นอุปกรณ์ “เครื่องปลายทางใบ้” (dumb terminal) ที่ใช้การควบคุมสายงานด้วยฮาร์ดแวร์ผ่านพินดีทีอาร์ (DTR : Data Terminal Ready) แทนพินอาร์ทีเอสเป็นต้น

การควบคุมสายงานด้วยซอฟต์แวร์จะทำโดยการส่งสัญญาณผ่านสายส่งข้อมูลโดยสัญญาณที่ส่งนั้นมี 2 คำสั่งคือ “เริ่ม” กับ “หยุด” โดยจะใช้ตัวอักษรควบคุมในรหัสแอสกี้ (ASCII control characters) คือดีซี 1 (DC1) สำหรับคำสั่งเริ่มและดีซี3 (DC3) สำหรับคำสั่งหยุดโดยการแทรกรหัสดังกล่าวเข้าไปในไบต์ข้อมูลขาส่งแต่เมื่อเทียบความเร็วของการควบคุมสายงานด้วยซอฟต์แวร์แล้วก็ยังถือว่าทำงานช้ากว่าการควบคุมสายงานด้วยฮาร์ดแวร์แม้ว่าการแทรกรหัสคำสั่งจะสามารถหยุดการส่งข้อมูลปกติได้ทั้งหมดเพื่อให้รหัสคำสั่งนี้ได้ส่งไปก่อนนอกจากนี้ยังต้องแก้ปัญหาในกรณีที่ในข้อมูลปกติมีไบต์ดีซี1 หรือไบต์ดีซี3 ซึ่งตรงกับรหัสคำสั่งโดยระบบต้องแยกให้ออกว่าไบต์ที่รับมานี้เป็นข้อมูลปกติหรือไบต์คำสั่ง

2.6.3.8 **เส้นทางกระแสข้อมูลและบัฟเฟอร์**

การรับส่งข้อมูลนั้นเกี่ยวข้องกับบัฟเฟอร์ 3 ส่วน (3 คู่บัฟเฟอร์ในกรณีรับและส่ง) คือไฟโฟ 16 ไบต์ในยูอาร์ต บัฟเฟอร์ใหญ่ที่อยู่ในอุปกรณ์ที่ต่อกับช่องทางข้อมูลอนุกรม (เช่นโมเด็ม) บัฟเฟอร์ใหญ่ 8,000 ไบต์ในหน่วยความจำหลักในคอมพิวเตอร์

เมื่อต้องการส่งไบต์ข้อมูลออกทางช่องทางข้อมูลอนุกรม โปรแกรมประยุกต์ต้องส่งไบต์ข้อมูลไปที่บัฟเฟอร์ใหญ่ขาส่ง 8,000 ไบต์ในหน่วยความจำหลักในคอมพิวเตอร์ส่วนบัฟเฟอร์ใหญ่ขารับ 8,000 ไบต์ในหน่วยความจำหลักก็จะทำหน้าที่ในทิศตรงกันข้ามกล่าวคือคอยรับข้อมูลที่ได้รับเข้ามา



รูป 2.7 การเคลื่อนไบต์ข้อมูลผ่าน 3 บัฟเฟอร์ที่อยู่ระหว่างโปรแกรมประยุกต์กับสายโทรศัพท์

กรณีส่งข้อมูลจากโปรแกรมประยุกต์ไปหาสายโทรศัพท์ (ซ้ายไปขวา) โปรแกรมประยุกต์ต้องส่งไบต์ข้อมูลไปที่บัฟเฟอร์ใหญ่ขาส่ง 8,000 ไบต์ในหน่วยความจำหลักในคอมพิวเตอร์จากนั้นโปรแกรมขับอุปกรณ์ของช่องทางข้อมูลอนุกรมจะย้ายไบต์ข้อมูลประมาณ 15 ไบต์จากบัฟเฟอร์ใหญ่ขาส่ง 8,000 ไบต์ไปให้กับไฟโฟ 16 ไบต์ในยูอาร์ตผ่านบัสข้อมูลทีละ 1 ไบต์เพื่อรอการส่งต่อให้เรจิสเตอร์ขาส่งทำการแปลงข้อมูลจากไบต์เป็นบิตและส่งออกทีละ 1 บิตออกทางพินขาส่งเชื่อมต่อไปยังสายเคเบิลซึ่งไบต์ข้อมูลที่มาถึงไฟโฟ 16 ไบต์แล้วจะต้องถูกส่งออกจากพินขาส่งทุกไบต์หลังจากนั้นบิตข้อมูลจะเคลื่อนไปตามสายเคเบิลเข้าสู่โมเด็ม (หรืออุปกรณ์อื่นใดที่เชื่อมต่อกับ

ช่องทางข้อมูลอนุกรม) ซึ่งมีบัฟเฟอร์ใหญ่ขนาด 1,000 ไบต์และหากบัฟเฟอร์ในโมเด็มเต็มทำให้ต้องส่งสัญญาณควบคุมสายงานมาบอกให้คอมพิวเตอร์หยุดส่งข้อมูลออกทางช่องทางข้อมูลอนุกรมซึ่งสัญญาณควบคุมสายงานจากโมเด็มนี้จะถูกตรวจสอบโดยโปรแกรมขับอุปกรณ์และเมื่อตรวจแล้วโปรแกรมขับอุปกรณ์จะสั่งหยุดการย้ายข้อมูลจากบัฟเฟอร์ใหญ่ขาส่ง 8,000 ไบต์ไปให้กับไฟโฟ 16 ไบต์ในยูอาร์ตแต่ถ้าบัฟเฟอร์ใหญ่ขาส่ง 8,000 ไบต์ยังมีที่ว่างและโปรแกรมประยุกต์ต้องการส่งข้อมูลผ่านช่องทางข้อมูลอนุกรมดังกล่าวข้อมูลที่จะส่งก็จะถูกนำมาเก็บไว้ในบัฟเฟอร์ใหญ่ขาส่ง 8,000 ไบต์และในเวลาเดียวกันนี้ข้อมูล 16 ไบต์ที่ค้างในไฟโฟในยูอาร์ตก็จะส่งออกไปทางพินขาส่งตามปกติ ส่วนโมเด็มก็จะส่งไบต์ข้อมูลออกไปตามสายโทรศัพท์จนมีที่ว่างในบัฟเฟอร์แล้วจึงส่งสัญญาณควบคุมสายงานให้คอมพิวเตอร์เริ่มส่งข้อมูลออกมาทางช่องทางข้อมูลอนุกรมมาที่โมเด็มได้ตามปกติต่อไป

ถ้าหากบัฟเฟอร์ใหญ่ขาส่ง 8,000 ไบต์มีข้อมูลอยู่เต็มบัฟเฟอร์แล้วโปรแกรมประยุกต์ที่จะส่งข้อมูลมาที่บัฟเฟอร์นี้จะถูกปฏิเสธซึ่งโปรแกรมประยุกต์จะหยุดการส่งชั่วคราวและรอจนกว่าบัฟเฟอร์ 8,000 ไบต์จะมีพื้นที่ว่างถึงจะเริ่มส่งข้อมูลเข้ามาได้ซึ่งในขณะที่รอนี้ซีพียูก็อาจจะสลับไปทำงานอื่น

จากที่กล่าวมายังมีข้อมูลเพิ่มเติมบางส่วนที่สรุปได้ 3 ประการคือประการแรกบิตข้อมูลที่อยู่ในไฟโฟจะต้องถูกส่งออกไปตามสายเคเบิลเสมอแต่อาจมียูอาร์ตบางตัวที่สามารถหยุดการส่งข้อมูลจากไฟโฟ 16 ไบต์ในยูอาร์ตไม่ให้ออกไปที่พินขาส่งได้แต่ลินุกซ์ยังไม่รองรับระบบนี้ ประการที่สองขณะที่โปรแกรมประยุกต์รอการส่งข้อมูลให้บัฟเฟอร์ใหญ่ขาส่ง 8,000 ไบต์นั้นซีพียูสามารถสลับไปทำงานอื่นได้ ประการที่สามโปรแกรมขับอุปกรณ์ของช่องทางข้อมูลอนุกรมจะมีบัฟเฟอร์ขนาดเล็กเป็นของตัวเองที่อยู่ในหน่วยความจำหลักเพื่อใช้จัดการกับไบต์ข้อมูล

2.6.3.9 **โปรแกรมขับช่องทางข้อมูลอนุกรม**

เป็นซอฟต์แวร์ที่ใช้จัดการงานที่เกี่ยวข้องกับช่องทางข้อมูลอนุกรมโดยโปรแกรมขับช่องทางข้อมูลอนุกรมนี้สามารถเลือกติดตั้งได้ 2 แบบคือแบบมอดูล (Module) และแบบในตัว (Built in) และตั้งแต่เคอร์เนล 2.2 เป็นต้นมาโปรแกรมขับที่ติดตั้งแบบมอดูลจะถูกโหลดโดยอัตโนมัติแต่ถ้าเป็นเคอร์เนลรุ่นก่อนหน้านั้นจะต้องสั่งดำเนินงาน “kerneld” เพื่อทำการโหลดมอดูลหรือมิฉะนั้นต้องประกาศใน /etc/modules ส่วนโปรแกรมขับช่องทางข้อมูลอนุกรมที่ติดตั้งแบบในตัวเข้าไปในเคอร์เนลแล้วสามารถใช้งานโปรแกรมขับได้ทันทีเมื่อต้องการโดยไม่ต้องโหลดโปรแกรมขับช่องทางข้อมูลอนุกรมแบบมอดูลมาซ้ำอีกเพราะจะทำให้เกิดความผิดพลาดขึ้นได้ขณะทำการเปิดช่องทางซึ่งสามารถตรวจมอดูลได้โดย “lsmod”

เมื่อโปรแกรมขับแบบมอดูลของช่องทางข้อมูลอนุกรมถูกโหลดมอนิเตอร์จะแสดงช่องทางข้อมูลอนุกรมที่มีอยู่ทั้งหมดพร้อมกับเลขไออาร์คิวโดย ณ จุดนี้เลขไออาร์คิวที่แสดงขึ้นมามักจะผิด แต่สามารถแก้ได้โดยใช้คำสั่ง “setserial” เพื่อเซตเลขไออาร์คิวที่ถูกต้องให้กับโปรแกรมขับช่องทางข้อมูลอนุกรมซึ่งในภายหลังก็จะแสดงช่องทางข้อมูลอนุกรมที่มีทั้งหมดพร้อมกับเลขไออาร์คิวที่ถูกต้อง

2.6.4 แผง/แผ่นวงจร/ตัวปรับต่อช่องทางข้อมูลอนุกรมหลายช่อง

2.6.4.1 บทนำช่องทางข้อมูลอนุกรมหลายช่อง

แผ่นวงจรช่องทางข้อมูลอนุกรมหลายช่องสามารถต่อใส่คอมพิวเตอร์ผ่านทางบัสไอเอสเอ (ISA Bus) หรือบัสพีซีไอ (PCI Bus) และนอกจากจะเรียก “...แผ่นวงจร” แล้วยังอาจเรียก “...แผง” หรือ “...ตัวปรับต่อ” โดยแผ่นวงจรนี้จะเพิ่มจำนวนช่องทางข้อมูลอนุกรมให้มากขึ้นซึ่งในปัจจุบันอาจใช้ในงานควบคุมที่ต้องการรับส่งข้อมูลกับอุปกรณ์หลาย ๆ ตัวในระยะทางใกล้ ๆ และใช้ความเร็วไม่มากนักและด้วยข้อจำกัดดังกล่าวแผ่นวงจรช่องทางข้อมูลอนุกรมหลายช่องจึงค่อย ๆ ลดความสำคัญลงและการนำมาใช้ก็ไม่แพร่หลายเหมือนอย่างที่เคยเป็นในอดีต

แผ่นวงจรหลายช่องบางแผ่นที่มีจำนวนหัวต่อมาก ๆ ในแผ่นวงจรเดียวกันเช่นดีบี 9 ดีบี 25 อาร์เจ 45 จะก่อให้เกิดปัญหาคือตัวแผ่นวงจรมีพื้นที่ไม่มากพอที่จะรองรับหัวต่อได้ทั้งหมดและเพื่อที่จะแก้ปัญหาดังกล่าวมักจะใช้สายเคเบิลพิเศษที่มีลักษณะคล้ายหนวดปลาหมึก (Octopus Cable) โดยใช้ปลายด้านหนึ่งต่อเข้ากับแผ่นวงจรส่วนปลายอีกด้านจะแบ่งออกไปต่อกับหัวต่อประเภทต่าง ๆ ที่อาจจะมีราวชั้นวาง (Rack) ให้หัวต่อเกาะซึ่งแยกต่างหากจากแผ่นวงจรก็เป็นได้

2.6.4.2 แผ่นวงจรช่องทางข้อมูลอนุกรมใบ้กับแผ่นวงจรช่องทางข้อมูลอนุกรมสมาร์ต

แผ่นวงจรช่องทางข้อมูลอนุกรมใบ้หลายช่องมีลักษณะไม่ต่างจากแผ่นวงจรช่องทางข้อมูลอนุกรมหลายช่องแบบเดิมมากนักเพราะงานเกือบทุกอย่างต้องอาศัยซีพียูเป็นผู้จัดการและโดยปกติแผ่นวงจรช่องทางข้อมูลอนุกรมใบ้หลายช่องจะใช้สัญญาณขัดจังหวะร่วมกันทั้งแผ่นวงจรซึ่งการทำเช่นนี้เมื่อซีพียูได้รับสัญญาณขัดจังหวะแล้วจะต้องตรวจสอบสถานะของช่องทางทุกช่องทางที่อยู่ในแผ่นวงจรช่องทางข้อมูลอนุกรมใบ้หลายช่องเพื่อหาว่าช่องทางใดร้องขอการบริการจากซีพียูเพื่อที่จะให้บริการได้อย่างถูกต้องต่อไปและแผ่นวงจรช่องทางข้อมูลอนุกรมใบ้หลายช่องบางตัวจะต้องใช้โปรแกรมขับเฉพาะของตัวเองเพื่อที่จะให้แผ่นวงจรทำงานได้

แผ่นวงจรสมาร์ตบางตัวอาจใช้ยูอาร์ตปกติทั่วไปแต่ภายในแผ่นวงจรสมาร์ตนั้นมักจะมีความสามารถจัดการกับสัญญาณขัดจังหวะจากยูอาร์ตได้เองโดยไม่ต้องอาศัยซีพียูแต่อย่างใดและในตัวแผ่นวงจรสมาร์ตก็จะมีบัฟเฟอร์ขนาดใหญ่ที่สามารถเก็บไบต์ข้อมูลไว้ก่อนที่จะส่งให้

บัฟเฟอร์ใหญ่ในหน่วยความจำหลักซึ่งการส่งแต่ละครั้งอาจส่งได้สูงถึง 1,000 ไบต์และแผ่นวงจรสมาร์ตบางตัวยังมีความสามารถรับส่งไบต์ข้อมูลผ่านบัสข้อมูลในคอมพิวเตอร์ได้เต็มความกว้างของบัสเช่นบัสข้อมูลที่กว้าง 64 บิตก็สามารถรับส่งได้ทีละ 64 บิตซึ่งต่างจากแผ่นวงจรช่องทางข้อมูลอนุกรมใบ้หลายช่องที่แม้ว่าบัสข้อมูลจะกว้างถึง 64 บิตแต่ก็รับส่งไบต์ข้อมูลผ่านบัสนี้ได้เพียงครั้งละ 8 บิต แต่แผ่นวงจรสมาร์ตก็มีหลายระดับแตกต่างกันกล่าวคือบางแผ่นวงจรก็ทำตามที่กล่าวได้ทั้งหมดแต่บางแผ่นวงจรก็ทำได้บางอย่างเท่านั้นซึ่งแผ่นวงจรสมาร์ตในปัจจุบันส่วนใหญ่จะเป็นแบบ “ต่อแล้วใช้” (PnP : plug-and-play)

2.6.4.3 โปรแกรมขับแผ่นวงจรหลายช่อง

เพื่อที่จะให้แผ่นวงจรหลายช่องทำงานได้จะต้องใช้โปรแกรมขับพิเศษของตัวแผ่นวงจรเองโดยโปรแกรมขับนี้ติดตั้งได้สองแบบคือแบบมอดูลและแบบในตัวซึ่งโดยทั่วไปแผ่นวงจรช่องทางข้อมูลอนุกรมใบ้มักนิยมติดตั้งโปรแกรมขับแบบในตัวส่วนแผ่นวงจรสมาร์ตนิยมติดตั้งโปรแกรมขับแบบมอดูล กรณีติดตั้งโปรแกรมขับแบบในตัวเคอร์เนล (นิยมใช้กับแผ่นวงจรช่องทางข้อมูลอนุกรมใบ้) สามารถทำได้โดยเลือกหัวข้อ "CONFIG_SERIAL_EXTENDED" รวมเข้าไปในเคอร์เนล (ตระกูล 2.4) ก่อนที่จะทำการแปลโปรแกรมเคอร์เนลส่วนกรณีติดตั้งโปรแกรมขับแบบมอดูล (นิยมใช้กับแผ่นวงจรสมาร์ต) สามารถทำได้โดยเลือกหัวข้อ "CONFIG_SERIAL_EXTENDED" ให้เป็นแบบมอดูลก่อนที่จะทำการแปลโปรแกรมเคอร์เนลซึ่งโปรแกรมขับแบบมอดูลจะถูกโหลดโดยอัตโนมัติเมื่อระบบต้องการใช้งานแผ่นวงจรสมาร์ต ส่วนเนื้อหาหรือข้อมูลของแผ่นวงจรแต่ละแผ่นวงจรนั้นสามารถหาได้จากเว็บไซต์ของผู้ผลิตแต่จะมีแผ่นวงจรบางแผ่นที่เคอร์เนลรองรับได้แน่นอนเช่น computone, hayes-esp, moxa-smartio, riscom8, specialix, stallion, and sx (specialix) ซึ่งข้อมูลของแผ่นวงจรเหล่านี้สามารถหาอ่านได้ในหัวข้อเอกสารภายในเคอร์เนล

2.6.4.4 แฟ้มช่องทางข้อมูลอนุกรมของแผ่นวงจรหลายช่องในสารบบ /dev

แฟ้มช่องทางข้อมูลอนุกรมที่อยู่ในแผ่นวงจรหลายช่องนั้นมีอยู่หลายชื่อแฟ้มขึ้นอยู่กับว่าใช้แผ่นวงจรชนิดไหนเพราะบางแผ่นวงจรจะใช้ชื่อเฉพาะของตัวเองเช่น /dev/ttyE0 (Stallion) หรือ /dev/ttyD0 (Digiboard) หรือ /dev/ttyF0 (Computone) เป็นต้น ส่วนชื่อมาตรฐานของช่องทางข้อมูลอนุกรมที่ใช้ทั่วไปคือ /dev/ttyS0 และชื่อแฟ้มเฉพาะของช่องทางข้อมูลอนุกรมของผู้ผลิตรายอื่นสามารถหาดูได้จาก “devices.txt” ในหัวข้อเอกสารภายในเคอร์เนล

2.6.4.4.1 **การสร้างแฟ้มช่องทางข้อมูลอนุกรมในสารบบ /dev**

ทำได้โดยใช้คำสั่ง “MAKEDEV” หรือ “mknod” ตัวอย่างเช่นต้องการสร้างแฟ้มช่องทางข้อมูลอนุกรมช่องทางที่ 1 โดยใช้ชื่อมาตรฐาน ttyS0 สามารถทำได้โดยคำสั่ง

```
[root]# MAKEDEV /dev/ttyS0
```

หรือ

```
[root]# mknod /dev/ttyS0 c 4 64
```

โดยคำสั่ง mknod นั้นจะต้องระบุชนิด (Type) เมเจอร์ (Major) ไมเนอร์ (Minor) ที่ถูกต้องด้วยโดยดูได้จาก “devices.txt” ในหัวข้อเอกสารภายในเคอร์เนล

2.6.4.5 **แผ่นวงจรช่องทางข้อมูลอนุกรมหลายช่องสำหรับคอมพิวเตอร์มาตรฐาน**

คอมพิวเตอร์ตั้งโต๊ะทั่ว ๆ ไปมักจะมีช่องทางข้อมูลอนุกรมมาให้ 1 หรือ 2 ช่องโดยช่องทางข้อมูลอนุกรมจะติดอยู่ด้านหลังแผงหลักของเครื่องคอมพิวเตอร์ซึ่งช่องทางข้อมูลอนุกรมที่ใช้ในปัจจุบันส่วนใหญ่จะใช้ยูอาร์ตรุ่น 16550 หรือรุ่น 16650 (บัฟเฟอร์ถาวร 32 ไบต์) และในกรณีที่ช่องทางข้อมูลอนุกรมที่มากับคอมพิวเตอร์มีไม่พอสำหรับการใช้งานก็สามารถซื้อแผ่นวงจรช่องทางข้อมูลอนุกรมมาต่อเพิ่มได้ (นิยมต่อกับบัสพีซีไอ) แต่ก่อนที่จะซื้อนั้นต้องตรวจสอบให้ชัดเจนว่าระบบที่ใช้งานอยู่ทั้งระบบปฏิบัติการและฮาร์ดแวร์สามารถทำงานเข้ากับแผ่นวงจรที่จะซื้อใหม่นั้นได้

2.6.4.6 **แผ่นวงจรช่องทางข้อมูลอนุกรมใบ้หลายช่อง**

หรืออาจเรียกว่า “ตัวปรับต่ออนุกรม” (Serial Adapter) โดยช่องทางแต่ละช่องของแผ่นวงจรจะต้องมีเลขที่อยู่เป็นของตัวเองและส่วนมากแผ่นวงจรเหล่านี้มักจะมีกรรมวิธีพิเศษให้แต่ละช่องทางใช้สัญญาณขัดจังหวะร่วมกันได้โดยต้องแปลโปรแกรมเคอร์เนลให้รองรับกรรมวิธีพิเศษดังกล่าว

2.6.4.7 **แผ่นวงจรช่องทางข้อมูลอนุกรมสมาร์ตหลายช่อง**

ก่อนจะซื้อหรือใช้แผ่นวงจรเหล่านี้ควรศึกษาข้อมูลให้แน่ใจว่าแผ่นวงจรดังกล่าวสามารถใช้งานเข้าได้กับระบบปฏิบัติการและฮาร์ดแวร์ที่ใช้อยู่หรือไม่เพราะแผ่นวงจรเหล่านี้มักจะต้องอาศัยโปรแกรมขับที่มากับแผ่นวงจรร่วมถึงอุปกรณ์พิเศษที่จะอยู่ใน /dev ที่ไม่ใช่ tts และข้อมูลของแผ่นวงจรเหล่านี้จะแตกต่างกันไปตามผู้ผลิตฮาร์ดแวร์

มอดูลของโปรแกรมขับในลินุกซ์จะใช้ชื่อ *.ko หรือไม่ก็ *.o ซึ่งโปรแกรมขับที่มากับเคอร์เนลนั้นอาจรองรับแผ่นวงจรบางแผ่นได้อยู่แล้วโดยไม่ต้องอาศัยโปรแกรมขับที่มากับแผ่นวงจร แต่อย่างไรก็ตามผู้ใช้ก็ต้องเซตค่าเลขที่อยู่ไอ/โอกับเลขไอออาร์คิวที่เหมาะสมให้กับโปรแกรมขับด้วย

2.6.4.8 **แผ่นวงจรหลายช่องที่ลินุกซ์ไม่รองรับ**

ตัวอย่างแผ่นวงจรที่ไม่ได้กล่าวถึงการใช้งานร่วมกับลินุกซ์ (1 มกราคม ค.ศ.2000)

- Aurora (PCI only) www.auroratech.com

2.6.5 **การเซตค่าพารามิเตอร์ให้ช่องทางข้อมูลอนุกรม**

ก่อนที่จะใช้งานช่องทางข้อมูลอนุกรมได้นั้นโปรแกรมขับจะต้องทราบพารามิเตอร์ของช่องทางก่อนเช่นชื่อ เลขที่อยู่ช่องทางไอ/โอ เลขไออาร์คิว ฯลฯ เป็นต้นและแม้ว่าจะไม่มีมาตรฐานกำหนดแน่นอนว่าช่องทางไหนจะต้องใช้เลขที่อยู่ไอ/โอกับเลขไออาร์คิวไหนแต่ในทางปฏิบัติแล้วช่องทางลำดับต้น ๆ (ประมาณ 4 ช่องทางแรก) จะมีค่าที่นิยมใช้งานกันและจะถูกกำหนดให้โปรแกรมขับใช้ค่าดังกล่าวได้เลยส่วนในกรณีที่ต้องการเปลี่ยนแปลงเป็นค่าอื่นก็สามารถทำได้โดยคำสั่งที่เหมาะสมในภายหลัง

การเซตระดับล่างเช่นการเซตเลขที่อยู่ไอ/โอและเลขไออาร์คิวให้ช่องทางจะต้องสอดคล้องกันทั้ง 2 ส่วนคือทางฮาร์ดแวร์และทางซอฟต์แวร์โดยทางฮาร์ดแวร์ทำโดยการเซตตัวโยงที่มักพบได้ในแผ่นวงจรบางรุ่นหรือการเซตที่ไบออสในกรรมวิธีแบบต่อแล้วใช้ ส่วนทางซอฟต์แวร์ทำผ่านคำสั่ง “setserial” เพื่อบอกโปรแกรมขับให้ทราบค่าเลขที่อยู่ไอ/โอและเลขไออาร์คิวที่ต้องการจะใช้

การเซตระดับบนเช่นการเซตความเร็ว (เช่น 9,600 บิต/วินาที) การเลือกใช้หรือไม่ใช้การควบคุมสายงานซึ่งปกติการเซตระดับบนสามารถทำได้ในตัวโปรแกรมสื่อสารเช่น PPP minicom getty หรือโดยโปรแกรม “stty”

2.6.6 **เลขที่อยู่ไอ/โอและเลขไออาร์คิวของช่องทางข้อมูลอนุกรม**

2.6.6.1 **บัสเชื่อมต่อระหว่างช่องทางข้อมูลอนุกรมกับคอมพิวเตอร์**

กรณีที่ช่องทางข้อมูลอนุกรมมาในรูปแบบแผ่นวงจรก็สามารถสังเกตจุดที่แผ่นวงจรเชื่อมต่อได้โดยง่ายซึ่งมักจะต่อกับคอมพิวเตอร์ผ่านบัสพีซีไอเป็นส่วนใหญ่แต่ถ้าช่องทางข้อมูลอนุกรมติดตั้งมาพร้อมกับแผงหลักการสังเกตอาจทำได้ยากแต่หากเป็นแผงหลักรุ่นเก่าแล้วแผงหลักจะต่อกับช่องทางข้อมูลอนุกรมทางบัสไอเอสเอเป็นหลักหรือไม่เช่นนั้นการเชื่อมต่ออาจเป็นบัสชนิดอื่นเช่นบัสแอลพีซี (LPC Bus) ซึ่งบัสชนิดหลังนี้ไม่มีมาตรฐานการเชื่อมต่อแบบต่อแล้วใช้

เพื่อที่จะเซตช่องทางระดับล่าง โดยหนทางเดียวที่จะเซตได้คือให้ไบออสของแผงหลักเซตผ่านเอซีพีไอ (ACPI) ให้เท่านั้น

2.6.6.2 **เลขที่อยู่ไอ/โอและเลขไออาร์คิว**

เพื่อจะให้ช่องทางข้อมูลอนุกรมทำงานได้ขั้นแรกต้องกำหนดเลขที่อยู่ไอ/โอและเลขไออาร์คิวที่ถูกต้องให้กับช่องทางนั้น ๆ สำหรับฮาร์ดแวร์รุ่นเก่า (รุ่นก่อนกลางปี ค.ศ.1990) การเซตค่าทางฮาร์ดแวร์ต้องอาศัยการเซตตัวโยงหรือใช้ค่าที่เซตไว้ในไบออสแต่สำหรับฮาร์ดแวร์รุ่นใหม่จะไม่มีตัวโยง การเซตค่าทางฮาร์ดแวร์จะทำแบบอัตโนมัติโดยไบออสหรือลินุกซ์ในช่วงการปลุกเครื่อง (Boot) ซึ่งไบออสหรือลินุกซ์จะส่งสัญญาณดิจิทัลไปสู่ฮาร์ดแวร์เพื่อเซตค่าดังกล่าวและในกรณีที่เซตเลขที่อยู่ไอ/โอผิดหรือช่องทางไม่มีเลขที่อยู่ไอ/โอเลยโปรแกรมขับจะไม่สามารถรับส่งข้อมูลผ่านช่องทางได้อย่างถูกต้องและถ้าช่องทางใช้เลขไออาร์คิวที่ผิดการรับส่งข้อมูลจะเป็นไปอย่างไม่มีประสิทธิภาพเพราะโปรแกรมขับต้องคอยวนมาตรวจสอบข้อมูลโดยกรรมวิธี “การหยั่งสัญญาณ” (Polling) ทำให้การรับส่งข้อมูลช้ามากจนถึงรับส่งข้อมูลไม่ได้เลย

โปรแกรมขับช่องทางข้อมูลอนุกรมจะต้องทราบเลขที่อยู่ไอ/โอและเลขไออาร์คิวเพื่อจะรับส่งข้อมูลผ่านชิปช่องทางข้อมูลอนุกรมโดยโปรแกรมขับรุ่นใหม่ ๆ (ตั้งแต่เคอร์เนล 2.4 เป็นต้นมา) จะพยายามหาเลขที่อยู่ไอ/โอและเลขไออาร์คิวด้วยกรรมวิธีแบบต่อแล้วใช้ทำให้ผู้ใช้ไม่ต้องเข้าไปเซตเอง (โดยคำสั่ง setserial) นอกจากนี้โปรแกรมขับรุ่นใหม่ ๆ อาจจะทำการเซตเลขที่อยู่ไอ/โอและเลขไออาร์คิวในทางฮารด์แวร์ให้อีกด้วย

2.6.6.3 **การเลือกไออาร์คิวให้ช่องทางข้อมูลอนุกรม**

ถ้าแผ่นวงจรช่องทางข้อมูลอนุกรมที่ใช้เป็นแบบต่อแล้วใช้แล้วการเซตค่าต่าง ๆ ให้กับช่องทางจะทำโดยซอฟต์แวร์ของกรรมวิธีต่อแล้วใช้ในไบออสหรือโดยโปรแกรมขับซึ่งซอฟต์แวร์ดังกล่าวจะหาค่าที่ซอฟต์แวร์คิดว่าเหมาะสมที่สุด (แต่อาจจะไม่ใช่ค่าที่ดีที่สุด) แล้วทำการเซตค่านั้นให้กับช่องทางโดยอัตโนมัติส่วนในกรณีที่ผู้ใช้ไม่ได้ใช้ช่องทางที่สามารถเซตค่าไออาร์คิวได้เองโดยอัตโนมัติและต้องเลือกไออาร์คิวเองในหัวข้อต่อไปจะกล่าวถึงการเลือกและการใช้งานไออาร์คิวต่าง ๆ รวมถึงแนะนำไออาร์คิว 0 ซึ่งมีการใช้งานที่พิเศษต่างจากไออาร์คิวอื่นทั่วไป

2.6.6.3.1 **ไออาร์คิว** 0

แท้จริงแล้วไออาร์คิว 0 เป็น “ตัวจับเวลา” (Timer) ในทางฮาร์ดแวร์แต่ไออาร์คิว 0 จะมีความหมายพิเศษเมื่อนำมาเซตในช่องทางข้อมูลอนุกรมด้วยคำสั่ง “setserial” โดยจะเป็นการบอกให้โปรแกรมขับทราบว่าช่องทางที่เซตไออาร์คิว 0 นี้ไม่มีหรือไม่ได้ใช้ไออาร์คิวแต่อย่างใดแต่การ

ทำงานกับช่องทางนี้จะใช้วิธีการหยั่งสัญญาณแทนซึ่งกรรมวิธีการหยั่งสัญญาณจะเป็นการเพิ่มงานให้ซีพียูโดยเฉพาะกรณีที่ช่องทางมีการรับส่งข้อมูลน้อยแต่ซีพียูต้องวนมาตรวจสอบตลอดเวลา แต่ทว่าไออาร์คิว 0 ก็มีประโยชน์ในการทดสอบช่องทางเช่นในกรณีที่ไออาร์คิวเกิดขัดกันหรือเซตไออาร์คิวผิดเพราะผู้ใช้สามารถใช้ไออาร์คิว 0 ได้โดยไม่จำเป็นต้องทราบการเซตไออาร์คิวหรือการขัดจังหวะทางฮาร์ดแวร์เลย แต่อย่างไรก็ตามไออาร์คิว 0 เป็นการใช้งานช่องทางด้วยการหยั่งสัญญาณซึ่งเหมาะใช้ในการทดสอบช่องทางเพียงชั่วคราวหรือจนกว่าจะสามารถหาค่าไออาร์คิวที่เหมาะสมได้ หลังจากนั้นควรจะเปลี่ยนไปใช้ไออาร์คิวที่เหมาะสมในการทำงานจริงต่อไป

2.6.6.3.2 การใช้สัญญาณขัดจังหวะร่วม (เคอร์เนล 2.2+)

คือมีอุปกรณ์ตั้งแต่ 2 อุปกรณ์ขึ้นไปใช้เลขไออาร์คิวซ้ำกันบนซีพียูตัวเดียวซึ่งโดยปกติจะทำได้บนบัสพีซีไอส่วนระบบที่ใช้บัสไอเอสเอหรือระบบที่ใช้บัสพีซีไอรวมกับบัสไอเอสเอจะไม่สามารถใช้สัญญาณขัดจังหวะร่วมได้เว้นแต่จะมีอุปกรณ์เสริมเพิ่มเติม แผ่นวงจรหลายช่องส่วนใหญ่จะใช้สัญญาณขัดจังหวะร่วมเพราะเลขสัญญาณขัดจังหวะมีอยู่อย่างจำกัดแต่การใช้สัญญาณขัดจังหวะร่วมก็มีข้อด้อยตรงที่เมื่อมีสัญญาณขัดจังหวะเกิดขึ้นแล้วระบบจะต้องตรวจว่าสัญญาณขัดจังหวะที่รับมานี้เกิดมาจากอุปกรณ์ตัวไหนซึ่งถ้าเป็นไปได้ควรใช้อินทอร์รัพแยกจะดีที่สุด

ตั้งแต่เคอร์เนล 2.2 เป็นต้นมาอาจมีการใช้สัญญาณขัดจังหวะร่วมกับช่องทางข้อมูลอนุกรมซึ่งการจะใช้สัญญาณขัดจังหวะร่วมได้นั้นเคอร์เนลจะต้องเลือกแปลโปรแกรม CONFIG_SERIAL_SHARE_IRQ เข้าไปในเคอร์เนลด้วย นอกจากนี้แล้วฮาร์ดแวร์ที่ใช้จะต้องรองรับการใช้สัญญาณขัดจังหวะร่วมได้เช่นกัน

2.6.6.3.3 การเลือกไออาร์คิว

ตาราง 2.4 ตัวอย่างฮาร์ดแวร์ไออาร์คิว 0 ถึง 15

IRQ	ASSIGNMENT
0	Timer channel 0 (May mean "no interrupt)
1	Keyboard
2	Cascade for controller 2
3	Serial port 2
4	Serial port 1
5	Parallel port 2, Sound card
6	Floppy diskette
7	Parallel port 1
8	Real−time clock
9	Redirected to IRQ2
10	not assigned
11	not assigned
12	not assigned
13	Math co−processor
14	Hard disk controller 1
15	Hard disk controller 2

ฮาร์ดแวร์ไออาร์คิวที่มีใช้อยู่ในปัจจุบันมีอยู่อย่างจำกัด ดังนั้นการเลือกไออาร์คิวจึงเป็นสิ่งสำคัญเพื่อป้องกันไม่ให้ไออาร์คิวเกิดขัดแย้งกันเองภายในระบบ พีซีปกติจะกำหนดให้ ttyS0 และ ttyS2 ใช้ไออาร์คิว 4 และให้ ttyS1 และ ttyS3 ใช้ไออาร์คิว 3 ส่วนไออาร์คิว 5 ใช้กับแผ่นวงจรเสียง แต่บ่อยครั้งที่ไออาร์คิว 5 ถูกนำมาใช้กับช่องทางข้อมูลอนุกรม

จะเห็นได้ว่าไม่มีคำตอบสำเร็จรูปในการเลือกไออาร์คิวหรือเลขการขัดจังหวะแต่การเลือกนั้นจะต้องหลีกเลี่ยงไม่ใช้ไออาร์คิวซ้ำกับแผงหลักหรือแผงอื่น ๆ ไออาร์คิวที่สามารถเลือกได้คือ 2 3 4 5 7 10 11 12 15 ซึ่งไออาร์คิว 2 กับไออาร์คิว 9 ทางโปรแกรมขับจะถือเป็นไออาร์คิวเดียวกัน ส่วนไออาร์คิวที่ไม่สามารถเลือกได้คือ 1 6 8 13 14 เพราะเป็นไออาร์คิวที่แผงหลักต้องใช้และภายหลังจากที่เลือกไออาร์คิวและเซตค่าแล้วควรตรวจความถูกต้องใน /proc/interrupts ระหว่างที่อุปกรณ์กำลังทำงานว่าไม่มีไออาร์คิวไหนทำงานขัดกัน

2.7 การโปรแกรมช่องทางข้อมูลอนุกรม (7)

2.7.1 บทนำการโปรแกรมช่องทางข้อมูลอนุกรม

ในบทนี้จะแนะนำการสื่อสารแบบอนุกรมบนคอมพิวเตอร์ด้วยมาตรฐานอาร์เอส-232 (RS-232) แต่อาจจะมีเนื้อหาบางส่วนคล้ายกับที่กล่าวมาแล้วในช่องทางข้อมูลอนุกรมแต่ในที่นี้จะมีรายละเอียดเพิ่มเติมเช่นสถานะของสัญญาณ การกำหนดสัญญาณพิน ฯลฯ รวมทั้งจะกล่าวถึงการเขียนโปรแกรมเพื่อใช้งานช่องทางอนุกรมด้วยภาษาซี

2.7.2 พื้นฐานการสื่อสารแบบอนุกรม

2.7.2.1 ความหมายทั่วไปของการสื่อสารแบบอนุกรม

โดยปกติคอมพิวเตอร์จะมีการรับส่งหรือเคลื่อนย้ายข้อมูลแบ่งได้สองประเภทใหญ่ ๆ ขึ้นกับจำนวนเส้นทางและจุดรับส่งกล่าวคือถ้าจำนวนเส้นทางและจุดรับส่งที่ใช้มีมากกว่าหนึ่งการรับส่งจะสามารถรับส่งพร้อมกันรวมแล้วได้มากกว่า 1 บิตซึ่งมักเรียกการรับส่งนี้ว่าการสื่อสารแบบขนานแต่ถ้าจำนวนเส้นทางและจุดรับส่งมีเพียงหนึ่งการรับส่งแบบนี้สามารถรับส่งข้อมูลได้ทีละ 1 บิตเท่านั้นและเรียกการรับส่งนี้ว่าการสื่อสารแบบอนุกรม

เมื่อกล่าวถึงการสื่อสารแบบอนุกรมระหว่างคอมพิวเตอร์กับอุปกรณ์ที่เชื่อมต่อนั้นข้อมูลที่รับส่งมักเป็นข้อมูลแบบดิจิตอลซึ่งโดยปกติแต่ละบิตนั้นมีได้หนึ่งในสองสถานะคือสถานะออน (on) กับสถานะออฟ (off) เท่านั้นซึ่งชื่อของสถานะอาจมีหลายชื่อแตกต่างกันไปเช่นมาร์ค (mark) แทนสถานะออนส่วนสเปซ (space) แทนสถานะออฟแต่อย่างไรก็ตามสรุปแล้วที่สภาวะปกติจะมีได้เพียงหนึ่งในสองสถานะตามที่กล่าวเท่านั้น

ความเร็วในการสื่อสารแบบอนุกรมมักวัดด้วยหน่วยบิตต่อวินาที (bps : bits-per-second) หรืออัตราบอด (baud : baudot rate) ซึ่งแสดงถึงจำนวนบิตทั้งหมดที่สามารถรับส่งได้ภายใน 1 วินาทีซึ่งในอดีตความเร็ว 300 บอดก็ถือว่ารวดเร็วมากในการรับส่งหรือการสื่อสารข้อมูล แต่ในปัจจุบันคอมพิวเตอร์สามารถรองรับความเร็วของอาร์เอส-232 ได้สูงถึง 430,800 บอด

เมื่อกล่าวถึงพอร์ตอนุกรมหรืออุปกรณ์ที่มีพอร์ตอนุกรมแล้วมักจะเรียกพอร์ตอนุกรมด้วยชื่อดีซีอี (DCE : Data Communications Equipment) หรือดีทีอี (DTE : Data Terminal Equipment) ตามการส่งและรับโดยลำดับ

2.7.2.2 ความหมายของอาร์เอส-232

อาร์เอส-232 เป็นมาตรฐานการเชื่อมต่อทางไฟฟ้าสำหรับการสื่อสารแบบอนุกรมซึ่งกำหนดโดยอีไอเอ (EIA : Electronic Industries Association) โดยอาร์เอส-232 นี้แบ่งได้ 3 แบบ คือเอ (A) บี (B) ซี (C) ซึ่งมีความแตกต่างกันที่ค่าแรงดันไฟฟ้าของสถานะออนกับสถานะออฟโดยแบบที่ได้รับความนิยมใช้งานมากคืออาร์เอส-232ซีโดยกำหนดให้สถานะออนหรือมาร์คมีค่าแรงดันไฟฟ้าอยู่ในช่วงระหว่าง -3 โวลต์ถึง -12 โวลต์ส่วนสถานะออฟหรือสเปซมีค่าแรงดันไฟฟ้าอยู่ในช่วงระหว่าง +3 โวลต์ถึง +12 โวลต์ โดยมาตรฐานอาร์เอส-232ซีที่ใช้แรงดันดังกล่าวทำให้การรับส่งข้อมูลสามารถทำระยะทางได้ถึง 25 ฟุต (8 เมตร) ก่อนที่จะรับส่งไม่ได้หรือมีข้อมูลผิดมากเกินกว่าที่จะรับได้แต่อย่างไรก็ตามระยะทางการรับส่งสามารถเพิ่มได้อีกหากลดอัตราบอดลงอย่างเหมาะสม

มาตรฐานการเชื่อมต่อแบบอนุกรมยังมีอีก 2 มาตรฐานคืออาร์เอส-422 (RS-422) และอาร์เอส-574 (RS-574) โดยอาร์เอส-422 จะใช้ค่าแรงดันไฟฟ้าที่ต่ำผนวกกับวิธีผลต่างสัญญาณทำให้สายสัญญาณมีระยะทางได้ไกลถึง 1000 ฟุต (300 เมตร) ส่วนอาร์เอส-574 ถูกกำหนดและออกแบบให้พอร์ตเป็นแบบ 9 พิน

2.7.2.3 สื่อสารสองทางเต็มอัตราและสื่อสารสองทางครึ่งอัตรา

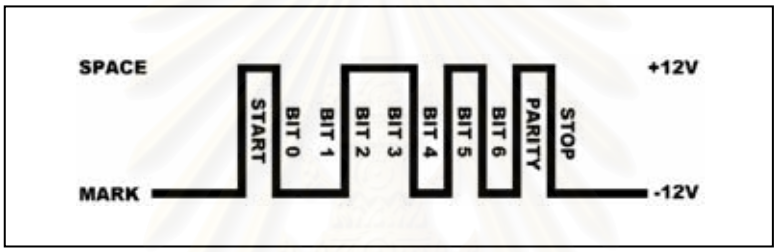
สื่อสารสองทางเต็มอัตราคือคอมพิวเตอร์สามารถรับและส่งข้อมูลได้ในเวลาเดียวกันจาก 2 ช่องสัญญาณคือ 1 ช่องรับและ 1 ช่องส่ง

สื่อสารสองทางครึ่งอัตราคือคอมพิวเตอร์ไม่สามารถรับและส่งข้อมูลได้ในเวลาเดียวกันเพราะการสื่อสารแบบนี้มักจะมีช่องสัญญาณเพียง 1 ช่องที่ทำหน้าที่ทั้งรับและส่งทำให้การรับและการส่งข้อมูลต้องทำต่างเวลากัน

2.7.2.4 การสื่อสารแบบอะซิงโครนัส

เพื่อที่จะให้คอมพิวเตอร์เข้าใจได้ว่ากำลังมีข้อมูลอนุกรมเข้ามาจะต้องมีสิ่งที่บ่งบอกจุดเริ่มต้นจุดปิดท้ายของอักขระ ซึ่งวิธีนี้เหมาะสำหรับการรับข้อมูลแบบอะซิงโครนัส

ในภาวะอะซิงโครนัสสายข้อมูลอนุกรมจะอยู่ในสถานะมาร์ค (1) ไปจนกว่าจะมีการส่งอักขระเข้ามาในสายข้อมูลและปกติทุก ๆ อักขระจะมีบิตพิเศษเพิ่มเข้ามาซึ่งเรียงลำดับบิตได้ดังนี้คือเริ่มต้นจากบิตเริ่ม (start bit) 1 บิตแล้วตามด้วยบิตอักขระ 7 หรือ 8 บิตแล้วตามด้วยบิตภาวะคู่หรือคี่ (parity bit) 0 หรือ 1 บิตที่เลือกได้แล้วปิดท้ายด้วยบิตหยุด (stop bit) 1 บิตหรือมากกว่า โดยบิตเริ่มจะอยู่ในสถานะสเปซ (0) เสมอเพื่อบอกคอมพิวเตอร์ว่ามีข้อมูลอนุกรมใหม่เข้ามาซึ่งการมีบิตพิเศษดังกล่าวทำให้การรับข้อมูลทำได้ทุกขณะเวลาใด ๆ และเรียกชื่อการสื่อสารแบบนี้ว่าเป็นแบบอะซิงโครนัส



รูป 2.8 รูปแบบไบต์ข้อมูลสำหรับการสื่อสารข้อมูลแบบอะซิงโครนัส

บิตภาวะคู่หรือคี่สามารถเลือกได้ว่าจะให้เป็นภาวะคู่หรือภาวะคี่ โดยบิตนี้จะต้องทำให้จำนวนรวมของบิต 1 เป็นจำนวนคู่ในกรณีที่เลือกให้เป็นบิตภาวะคู่ แต่ในทางตรงข้ามบิตนี้จะต้องทำให้จำนวนรวมของบิต 1 เป็นจำนวนคี่ในกรณีที่เลือกให้เป็นบิตภาวะคี่ ซึ่งบิตภาวะคู่หรือคี่ดังกล่าวจะต้องใช้บิต 1 บิตเพื่อทำให้เงื่อนไขเป็นจริงแต่ยังมีตัวเลือกอื่นอีกเช่นอาจจะไม่ใช้บิตภาวะคู่หรือคี่เลยก็เป็นได้

ส่วนบิตหยุดที่เป็นบิตปิดท้ายอักขระสามารถมีได้ 1 บิต 1.5 บิตหรือ 2 บิต ซึ่งบิตหยุดนี้จะอยู่ในสถานะมาร์ค (1) เสมอเพื่อสร้างระยะห่างระหว่างอักขระไม่ให้ใกล้กันจนเกินไปและให้เวลาคอมพิวเตอร์จัดการกับอักขระก่อนหน้า

รูปแบบข้อมูลอะซิงโครนัสนิยมเขียนเป็น 3 ส่วนคือบิตข้อมูล (7, 8) บิตภาวะคู่หรือคี่ (E, O, N) บิตหยุด (1, 1.5, 2) เช่น 8N1 คือ 8 บิตข้อมูล ไม่มีบิตภาวะคู่หรือคี่ 1 บิตหยุด ส่วน 7E1 คือ 7 บิตข้อมูล บิตภาวะคู่ 1บิตหยุด

2.7.2.5 การสื่อสารแบบซิงโครนัส

ต่างกับการสื่อสารแบบอะซิงโครนัสตรงที่ข้อมูลรับส่งมักจะมีลักษณะเป็นกระแสบิตข้อมูลแบบคงที่และการรับส่งข้อมูลระหว่างผู้รับและผู้ส่งจะต้องประสานเวลากันโดยใช้สัญญาณนาฬิกาเดียวกันหรือไม่เช่นนั้นสัญญาณนาฬิกาก็ต้องตรงกันเพื่อให้เกิดการซิงโครนัส

แม้ว่าจะการสื่อสารแบบซิงโครนัสจะมีเวลารับส่งที่ตรงกันแต่คอมพิวเตอร์ก็ยังต้องมีเครื่องหมายกำกับจุดเริ่มต้นของข้อมูลด้วยวิธีการบางวิธี แต่วิธีที่ได้รับความนิยมนำมาใช้งานคือใช้โพรโทคอลกลุ่มข้อมูลเช่น Serial Data Link Control (SDLC) หรือ High-Speed Data Link Control (HDLC) ซึ่งแต่ละโพรโทคอลจะกำหนดลำดับบิตที่แน่นอนเพื่อใช้แทนจุดเริ่มต้นและจุดปิดท้ายของกลุ่มข้อมูลรวมทั้งจะกำหนดลำดับบิตที่แสดงว่าไม่มีข้อมูลที่จะส่งเป็นผลให้ฝ่ายรับสามารถรับและแยกแยะข้อมูลได้อย่างถูกต้องโดยสังเกตจากจุดเริ่มและจุดปิดท้ายของกลุ่ม

เหตุที่โพรโทคอลแบบซิงโครนัสไม่ได้ใช้บิตซิงโครนัสต่ออักขระทำให้ประสิทธิภาพของการสื่อสารแบบซิงโครนัสดีขึ้นมากกว่า 25% เมื่อเทียบกับการสื่อสารแบบอะซิงโครนัส ซึ่งการสื่อสารแบบซิงโครนัสนี้เหมาะสำหรับการสื่อสารระหว่างเครือข่ายระยะไกลและมีส่วนต่อประสานอนุกรมตั้งแต่ 2 ส่วนขึ้นไป และแม้การสื่อสารแบบซิงโครนัสจะมีข้อได้เปรียบด้านความเร็วแต่ฮาร์ดแวร์อาร์เอส-232 ส่วนใหญ่กลับไม่รองรับการสื่อสารระบบดังกล่าวเพราะต้องใช้ทั้งฮาร์ดแวร์และซอฟต์แวร์พิเศษเพิ่มเติม

2.7.2.6 การเข้าถึงช่องทางข้อมูลอนุกรม

เช่นเดียวกับอุปกรณ์อื่นทั่วไป ยูนิกซ์ใช้การเข้าถึงช่องทางข้อมูลอนุกรมผ่านแฟ้มอุปกรณ์ (device file) และเพื่อที่จะเข้าถึงช่องทางข้อมูลอนุกรมทางผู้ใช้จะต้องเปิดแฟ้มอุปกรณ์ที่สอดคล้องกับช่องทางข้อมูลอนุกรมที่ต้องการเข้าถึง

2.7.2.7 แฟ้มช่องทางข้อมูลอนุกรม

ช่องทางข้อมูลอนุกรมในยูนิกซ์แต่ละช่องทางต่างมีแฟ้มอุปกรณ์อยู่ในสารบบ /dev ดังนี้

ตาราง 2.5 แฟ้มอุปกรณ์ช่องทางข้อมูลอนุกรม

System	Port 1	Port 2
IRIX®	/dev/ttyf1	dev/ttyf2
HP–UX	/dev/tty1p0	dev/tty2p0
Solaris®/SunOS®	/dev/ttya	dev/ttyb
Linux®	/dev/ttyS0	dev/ttyS1
Digital UNIX®	/dev/tty01	dev/tty02

2.7.2.8 การเปิดช่องทางข้อมูลอนุกรม

เมื่อช่องทางข้อมูลอนุกรมมีสถานะเป็นแฟ้ม การจะเข้าถึงแฟ้มจะต้องได้รับอนุญาตก่อน จึงจะเปิดแฟ้มได้ โดยการเปิดแฟ้มก็ใช้ฟังก์ชัน open() เช่นเดียวกับที่ใช้เปิดแฟ้มทั่ว ๆ ไป

2.7.2.8.1 ตัวเลือกการเปิด

ตัวบ่งชี้ที่อยู่ในฟังก์ชัน open() ถือเป็นตัวเลือกของการเปิดแฟ้มให้ทำงานเป็นไปตามที่ต้องการโดย

- ตัวบ่งชี้ O_RDWR เพื่อระบุว่าการเปิดแฟ้มครั้งนี้สามารถอ่านข้อมูลจากแฟ้มและเขียนข้อมูลลงแฟ้มได้

- ตัวบ่งชี้ O_NOCTTY เพื่อระบุว่าโปรแกรมที่เปิดแฟ้มนี้ไม่ต้องการให้ถูกควบคุมจากเครื่องปลายทางสำหรับช่องทางเพราะถ้าไม่ระบุตัวบ่งชี้นี้เมื่อมีอินพุตใด ๆ เข้ามาเช่นมีสัญญาณ abort จากแผงแป้นอักขระเข้ามาก็จะส่งผลกระทบกับกระบวนการนี้ทันที

- ตัวบ่งชี้ O_NDELAY เพื่อระบุว่าโปรแกรมที่เปิดแฟ้มนี้ไม่สนใจสถานะของสัญญาณดีซีดีที่ใช้แสดงสถานะการทำงานของอุปกรณ์ปลายทางว่ากำลังทำงานอยู่หรือไม่ ในกรณีที่ไม่ได้ระบุตัวบ่งชี้นี้ในฟังก์ชัน open() กระบวนการจะหลับไปจนกว่าจะมีแรงดันสเปซบนสายสัญญาณดีซีดี

2.7.2.9 เขียนข้อมูลไปช่องทาง

การเขียนข้อมูลไปยังช่องทางสามารถทำได้โดยง่ายเพียงอาศัยฟังก์ชัน write() เพื่อที่จะส่งข้อมูลออกไป

```
n = write(fd, "ATZ\r", 4);
if (n < 0)
fputs("write() of 4 bytes failed!\n", stderr);
```

2.7.2.10 อ่านข้อมูลจากช่องทาง

การอ่านข้อมูลจากช่องทางมีกรรมวิธีที่ซับซ้อนกว่าการเขียนเล็กน้อยเพราะต้องเซตตัวเลือกให้การรับเป็นไปตามความต้องการเพราะการรับจะมีลักษณะการทำงานหลายภาวะหลายแบบเช่นช่องทางที่ทำงานในภาวะข้อมูลดิบ (raw data mode) ฟังก์ชัน read() จะรีเทิร์นจำนวนอักขระที่อยู่ในอินพุตบัฟเฟอร์ของช่องทางทันทีแต่ถ้าในอินพุตบัฟเฟอร์ไม่มีอักขระอยู่เลยฟังก์ชัน read() จะถูกบล็อกไปจนกว่าจะมีอักขระเข้ามาหรือหมดเวลารอหรือมีความผิดพลาดเกิดขึ้น

ฟังก์ชัน read() สามารถกำหนดให้รีเทิร์นได้ทันทีด้วยคำสั่ง fcntl(fd, F_SETFL, FNDELAY) เพราะตัวบ่งชี้ FNDELAY จะทำให้ฟังก์ชัน read() รีเทิร์น 0 ทันทีในกรณีที่ไม่มีอักขระอยู่ในบัฟเฟอร์ แต่หากมีความต้องการกลับไปใช้งานตามปกติคือให้ฟังก์ชัน read() ถูกบล็อกเมื่อไม่มีอักขระอยู่ในบัฟเฟอร์ก็สามารถทำได้ด้วยคำสั่ง fcntl(fd, F_SETFL, 0)

2.7.2.11 **การปิดช่องทาง**

การปิดช่องทางสามารถทำได้โดยฟังชัน close() และภายหลังจากที่ปิดช่องทางแล้ว สัญญาณดีทีอาร์จะถูกเซตให้มีแรงดันมาร์คเพื่อแจ้งให้อุปกรณ์ปลายทางทราบว่าอุปกรณ์นี้ไม่อยู่ในสถานะที่จะรับส่งข้อมูลผ่านช่องทางนี้แล้ว

2.7.3 **โครงแบบช่องทางข้อมูลอนุกรม**

2.7.3.1 **ส่วนต่อประสานเครื่องปลายทางโพสิกซ์**

ระบบส่วนใหญ่ในปัจจุบันมักจะรองรับส่วนต่อประสานเครื่องปลายทางโพสิกซ์ได้เพื่อใช้ดัดแปรพารามิเตอร์ต่าง ๆ เช่นอัตราบอด ขนาดอักขระ ฯลฯ โดยจะต้องประกาศ #include <termios.h> เพื่อระบุโครงสร้างที่ใช้ควบคุม โดยฟังก์ชันโพสิกซ์ 2 ฟังก์ชันสำคัญที่ใช้เปลี่ยนพารามิเตอร์ได้แก่ tcgetattr() และ tcsetattr() ซึ่งจะทำหน้าที่อ่านและเซตลักษณะประจำของโครงแบบตามลำดับ การเปลี่ยนโครงแบบจะต้องใช้ตัวชี้ให้ชี้ไปยังโครงสร้าง termios ที่มีสมาชิกดังนี้

ตาราง 2.6 สมาชิกโครงสร้าง termios

Member	Description
c_cflag	Control options
c_lflag	Line options
c_iflag	Input options
c_oflag	Output options
c_cc	Control characters
c_ispeed	Input baud (new interface)
c_ospeed	Output baud (new interface)

2.7.3.2 **ตัวเลือกควบคุม**

สมาชิก c_cflag นี้จะใช้ควบคุมอัตราบอด จำนวนบิตข้อมูล ภาวะคู่หรือคี่ บิตหยุด การควบคุมสายงานด้วยฮาร์ดแวร์ ฯลฯ ด้วยการเลือกตัวเลือกตามตาราง 2.7

สมาชิก c_cflag นิยมจะเปิดทาง 2 ตัวเลือกสำคัญคือ CLOCAL เพื่อให้แน่ใจได้ว่าช่องทางจะไม่มีการเปลี่ยนเจ้าของในขณะเปิดใช้งานและ CREAD เพื่อให้ช่องทางรับข้อมูลได้

ส่วนอัตราบอดที่เป็นตัวเลือกในสมาชิก c_cflag เป็นการเซตที่ใช้สำหรับส่วนต่อประสานแบบดั้งเดิมเพราะการเซตแบบนี้อัตราบอดรับและส่งจะมีค่าเท่ากัน สำหรับผู้ที่ต้องการเซตอัตรา

บอดรับค่าหนึ่งและอัตราบอดส่งอีกค่าหนึ่งสามารถทำได้ในสมาชิก c_ispeed และ c_ospeed ตามลำดับ

การเลือกตัวเลือกให้กับสมาชิก c_cflag นิยมใช้ตัวดำเนินการแอนด์ (AND) ออร์ (OR) นอต (NOT) เพื่อที่จะเซตหรือลบล้างบิตตัวเลือกเพราะการเพิ่มหรือลดตัวเลือกจากชุดตัวเลือกเดิมที่มีอยู่จะป้องกันไม่ให้ตัวเลือกที่โปรแกรมขับต้องการใช้ได้รับผลกระทบ

ตาราง 2.7 ตัวเลือกของสมาชิก c_cflag

Constant	Description
CBAUD	Bit mask for baud rate
B0	0 baud (drop DTR)
B50	50 baud
B75	75 baud
B110	110 baud
B134	134.5 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud
B2400	2400 baud
B4800	4800 baud
B9600	9600 baud
B19200	19200 baud
B38400	38400 baud
B57600	57,600 baud
B76800	76,800 baud
B115200	115,200 baud
EXTA	External rate clock
EXTB	External rate clock
CSIZE	Bit mask for data bits
CS5	5 data bits
CS6	6 data bits
CS7	7 data bits
CS8	8 data bits
CSTOPB	2 stop bits (1 otherwise)
CREAD	Enable receiver
PARENB	Enable parity bit
PARODD	Use odd parity instead of even
HUPCL	Hangup (drop DTR) on last close
CLOCAL	Local line – do not change "owner" of port
LOBLK	Block job control output
CNEW_RTSCTS CRTSCTS	Enable hardware flow control (not supported on all platforms)

2.7.3.2.1 **การเซตอัตราบอด**

อัตราบอดจะถูกเก็บในที่ต่างกันขึ้นกับระบบปฏิบัติการโดยส่วนต่อประสานแบบเดิมจะเก็บอัตราบอดไว้ในสมาชิก c_cflag ซึ่งใช้อัตราบอดได้ตามที่กำหนดในตาราง 2.7 ขณะที่ส่วนต่อประสานแบบใหม่จะมีสมาชิก c_ispeed และ c_ospeed เก็บอัตราบอดรับและส่งแยกกัน ซึ่งการเซตอัตราบอดในโครงสร้าง termios ให้กับสมาชิกทั้ง 2 ทำได้โดยฟังก์ชัน cfsetispeed() และ cfsetospeed() ตามลำดับ

2.7.3.2.2 การเซตขนาดอักขระ

การเซตขนาดอักขระสามารถเซตได้ตามค่าที่กำหนดในตาราง 2.7 โดยมีขั้นตอนดังนี้

```
options.c_cflag &= ~CSIZE; /* Mask the character size bits */
options.c_cflag |= CS8; /* Select 8 data bits */
```

2.7.3.2.3 การเซตภาวะคู่หรือคี่

สามารถเซตได้ 3 แบบคือภาวะคู่ ภาวะคี่ ไม่มีทั้งภาวะคู่หรือคี่

```
/* No parity (8N1): */
options.c_cflag &= ~PARENB
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;

/* Even parity (7E1): */
options.c_cflag |= PARENB
options.c_cflag &= ~PARODD
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS7;

/* Odd parity (7O1): */
options.c_cflag |= PARENB
options.c_cflag |= PARODD
options.c_cflag &= ~CSTOPB
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS7;

/* Space parity is setup the same as no parity (8S1): */
options.c_cflag &= ~PARENB
options.c_cflag &= ~CSTOPB
```

```
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;
```

2.7.3.2.4 การเซตการควบคุมสายงานด้วยฮาร์ดแวร์

ยูนิกซ์บางรุ่นใช้การควบคุมสายงานด้วยฮาร์ดแวร์ผ่านสายสัญญาณซีทีเอสและอาร์ทีเอส ซึ่งการจะใช้งานดังกล่าวได้จะต้องระบุตัวเลือก CNEW_RTSCTS หรือ CRTSCTS ดังนี้

```
options.c_cflag |= CNEW_RTSCTS; /* Also called CRTSCTS */
```

ถ้าหากต้องการปิดทางการควบคุมสายงานด้วยฮาร์ดแวร์

```
options.c_cflag &= ~CNEW_RTSCTS;
```

2.7.3.3 ตัวเลือกท้องถิ่น

สมาชิก c_lflag ใช้บอกโปรแกรมขับช่องทางข้อมูลอนุกรมว่าจะให้โปรแกรมจัดการกับอักขระที่รับเข้ามาอย่างไรโดยมีตัวเลือกดังต่อไปนี้

ตาราง 2.8 ตัวเลือกของสมาชิก c_lflag

Constant	Description
ISIG	Enable SIGINTR, SIGSUSP, SIGDSUSP, and SIGQUIT signals
ICANON	Enable canonical input (else raw)
XCASE	Map uppercase \lowercase (obsolete)
ECHO	Enable echoing of input characters
ECHOE	Echo erase character as BS–SP–BS
ECHOK	Echo NL after kill character
ECHONL	Echo NL
NOFLSH	Disable flushing of input buffers after interrupt or quit characters
IEXTEN	Enable extended functions
ECHOCTL	Echo control characters as ^char and delete as ~?
ECHOPRT	Echo erased character as character erased
ECHOKE	BS–SP–BS entire line on line kill
FLUSHO	Output being flushed
PENDIN	Retype pending input at next read or input char
TOSTOP	Send SIGTTOU for background output

2.7.3.3.1 อินพุตคานอนนิแคล

เป็นการจัดการกับอักขระที่รับเข้ามาในแบบที่อาจเรียกว่าจัดการทีละบรรทัดเพราะอักขระที่รับเข้ามาจะถูกนำไปเก็บในบัฟเฟอร์แต่ทว่าผู้รับจะยังไม่สามารถเข้าถึงอักขระกลุ่มนี้ได้จนกว่าจะได้รับอักขระ CR (carriage return) หรือ LF (line feed) เป็นการปิดท้ายเพื่อบอกว่าจบบรรทัดแล้วเท่านั้น สำหรับการเซตใช้งานภาวะเช่นนี้จะต้องเลือกตัวเลือก ICANON, ECHO, ECHOE ดังนี้

```
options.c_lflag |= (ICANON | ECHO | ECHOE);
```

2.7.3.3.2 **อินพุตดิบ**

อักขระที่รับเข้ามาจะถูกนำไปเก็บในบัฟเฟอร์โดยผู้รับสามารถเข้าถึงอักขระที่ได้รับในทันทีแบบอักขระต่ออักขระ สำหรับการเซตใช้งานภาวะนี้จะต้องไม่เลือกตัวเลือก ICANON, ECHO, ECHOE, ISIG ดังนี้

```
options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
```

สำหรับค่าคงที่ ECHO* จะต้องถูกเลือกให้ปิดทางเมื่อต้องการส่งคำสั่งไปให้โมเด็มหรือคอมพิวเตอร์อื่นเพื่อป้องกันผลป้อนกลับวนซ้ำที่จะเกิดระหว่างส่วนต่อประสานอนุกรมทั้ง 2 ฝัง

2.7.3.4 **ตัวเลือกอินพุต**

สมาชิก c_iflag ใช้ควบคุมกระบวนการที่ใช้จัดการกับอักขระที่ได้รับเข้ามาจากช่องทางข้อมูลอนุกรม โดยมีตัวเลือกดังต่อไปนี้

ตาราง 2.9 ตัวเลือกของสมาชิก c_iflag

Constant	Description
INPCK	Enable parity check
IGNPAR	Ignore parity errors
PARMRK	Mark parity errors
ISTRIP	Strip parity bits
IXON	Enable software flow control (outgoing)
IXOFF	Enable software flow control (incoming)
IXANY	Allow any character to start flow again
IGNBRK	Ignore break condition
BRKINT	Send a SIGINT when a break condition is detected
INLCR	Map NL to CR
IGNCR	Ignore CR
ICRNL	Map CR to NL
IUCLC	Map uppercase to lowercase
IMAXBEL	Echo BEL on input line too long

2.7.3.4.1 **การเซตตัวเลือกภาวะคู่หรือคี่ของอินพุต**

ผู้รับสามารถเลือกตรวจสอบบิตภาวะคู่หรือคี่ได้ในกรณีอักขระที่รับมามีบิตภาวะคู่หรือคี่กำกับความถูกต้องมาด้วยโดยมีตัวเลือกที่เกี่ยวข้องคือ INPCK, IGNPAR, PARMRK, ISTRIP ซึ่งการเลือกตรวจสอบบิตภาวะคู่หรือคี่พร้อมทั้งถอดบิตดังกล่าวออกทำได้ดังนี้

```
options.c_iflag |= (INPCK | ISTRIP);
```

ตัวเลือก IGNPAR ใช้บอกให้โปรแกรมขับไม่สนใจความผิดพลาดบิตภาวะคู่หรือคี่โดยจะส่งผ่านอักขระที่รับเข้ามาให้ผู้รับเสมือนหนึ่งไม่มีความผิดพลาดเกิดขึ้น

ตัวเลือก PARMRK ใช้ทำเครื่องหมายนำหน้าให้กับอักขระที่มีบิตภาวะคู่หรือคี่ผิดพลาด ในกรณีที่เปิดทาง IGNPAR แล้วเครื่องหมายนำหน้าอักขระที่มีบิตภาวะคู่หรือคี่ผิดพลาดจะใช้อักขระ NUL

ส่วนกรณีอื่นเครื่องหมายนำหน้าอักขระที่มีบิตภาวะคู่หรือคี่ผิดพลาดจะใช้อักขระ DEL และอักขระ NUL

2.7.3.4.2 การเซตการควบคุมสายงานด้วยซอฟต์แวร์

การเปิดทางการควบคุมสายงานด้วยซอฟต์แวร์มีตัวเลือกที่ต้องเปิดใช้คือ IXON, IXOFF, IXANY

```
options.c_iflag |= (IXON | IXOFF | IXANY);
```

แต่หากต้องการปิดทางการควบคุมสายงานด้วยซอฟต์แวร์ก็ทำได้ดังนี้

```
options.c_iflag &= ~(IXON | IXOFF | IXANY);
```

โดย IXON แทนอักขระเริ่มข้อมูลและ XOFF แทนอักขระหยุดข้อมูลตามที่ได้กำหนดในสมาชิก c_cc

2.7.3.5 ตัวเลือกเอาต์พุต

สมาชิก c_oflag ใช้ควบคุมกระบวนการที่ใช้จัดการกับอักขระที่จะส่งออกไปตามช่องทางข้อมูลอนุกรม โดยมีตัวเลือกดังต่อไปนี้

ตาราง 2.10 ตัวเลือกของสมาชิก c_oflag

Constant	Description
OPOST	Postprocess output (not set = raw output)
OLCUC	Map lowercase to uppercase
ONLCR	Map NL to CR–NL
OCRNL	Map CR to NL
NOCR	No CR output at column 0
ONLRET	NL performs CR function
OFILL	Use fill characters for delay
OFDEL	Fill character is DEL
NLDLY	Mask for delay time needed between lines
NL0	No delay for NLs
NL1	Delay further output after newline for 100 milliseconds
CRDLY	Mask for delay time needed to return carriage to left column
CR0	No delay for CRs
CR1	Delay after CRs depending on current column position
CR2	Delay 100 milliseconds after sending CRs
CR3	Delay 150 milliseconds after sending CRs
TABDLY	Mask for delay time needed after TABs
TAB0	No delay for TABs
TAB1	Delay after TABs depending on current column position
TAB2	Delay 100 milliseconds after sending TABs
TAB3	Expand TAB characters to spaces
BSDLY	Mask for delay time needed after BSs
BS0	No delay for BSs
BS1	Delay 50 milliseconds after sending BSs
VTDLY	Mask for delay time needed after VTs
VT0	No delay for VTs
VT1	Delay 2 seconds after sending VTs
FFDLY	Mask for delay time needed after FFs
FF0	No delay for FFs
FF1	Delay 2 seconds after sending FFs

2.7.3.5.1 เอาต์พุตประมวลผล

การประมวลผลเอาต์พุตสามารถเซตได้ด้วยการเปิดทางตัวเลือก OPOST ที่อยู่ในสมาชิก c_oflag ดังนี้

```
options.c_oflag |= OPOST;
```

2.7.3.5.2 เอาต์พุตดิบ

เซตได้ด้วยการปิดทางตัวเลือก OPOST ที่อยู่ในสมาชิก c_oflag ดังนี้

```
options.c_oflag &= ~OPOST;
```

เมื่อตัวเลือก OPOST ถูกปิดทางแล้วตัวเลือกอื่นในสมาชิก c_oflag จะอยู่ในสถานะถูกเมินหรือไม่สนใจทันที

2.7.3.6 อักขระควบคุม

สมาชิก c_cc ใช้ควบคุมบทนิยามอักขระต่าง ๆ เช่นพารามิเตอร์หมดเวลารอ โดยสมาชิก c_cc มีตัวเลือกดังนี้

ตาราง 2.11 ตัวเลือกของสมาชิก c_cc

Constant	Description	Key
VINTR	Interrupt	CTRL–C
VQUIT	Quit	CTRL–Z
VERASE	Erase	Backspace (BS)
VKILL	Kill–line	CTRL–U
VEOF	End–of–file	CTRL–D
VEOL	End–of–line	Carriage return (CR)
VEOL2	Second end–of–line	Line feed (LF)
VMIN	Minimum number of characters to read	–
VSTART	Start flow	CTRL–Q (XON)
VSTOP	Stop flow	CTRL–S (XOFF)
VTIME	Time to wait for data (tenths of seconds)	–

2.7.3.6.1 การเซตอักขระการควบคุมสายงานด้วยซอฟต์แวร์

ตัวเลือก VSTART และ VSTOP จะเก็บอักขระที่ใช้สำหรับการควบคุมสายงานด้วยซอฟต์แวร์ โดยปกติตัวเลือกทั้ง 2 จะถูกเซตให้เป็นอักขระ DC1 (0x11) และ DC3 (0x13) ตามมาตรฐานแอสกีเพื่อแทน XON และ XOFF ตามลำดับ

2.7.3.6.2 การเซตหมดเวลารอการอ่าน

โปรแกรมขับส่วนต่อประสานอนุกรมมีความสามารถที่จะระบุให้การอ่านรอข้อมูลได้ไม่เกินระยะเวลาที่กำหนดโดยเรียกระยะเวลานี้ว่าหมดเวลารอ ตัวเลือกภายในสมาชิก c_cc ที่ใช้กำหนดเวลารอได้แก่ VMIN และ VTIME แต่การเซตหมดเวลารอจะถูกเมินถ้าการรับข้อมูลอยู่ใน

ภาวะอินพุตคานอนนิแคลหรือมีการเซตตัวเลือก NDELAY บนแฟ้มผ่านฟังก์ชัน open() หรือ fcntl()

VMIN ใช้ระบุจำนวนอักขระต่ำสุดที่จะอ่าน ในกรณีเซต VMIN เป็น 0 VTIME จะระบุเวลาที่รออ่านอักขระก่อนจะสิ้นสุดการอ่านส่วนในกรณี VMIN ไม่เป็น 0 VTIME จะระบุเวลาที่รออ่านอักขระแรกและหากอ่านอักขระแรกได้ภายในระยะเวลา VTIME ที่กำหนดจะทำให้การอ่านถูกบล็อกไปจนกว่าจะอ่านได้ครบจำนวน VMIN แต่หากอ่านอักขระแรกไม่ได้ภายในระยะเวลา VTIME ที่กำหนดจะถือว่าอ่านได้ 0 อักขระซึ่งกรรมวิธีเช่นนี้เป็นการบอกโปรแกรมขับว่าการอ่านครั้งนี้มีผลได้ 2 แบบคืออ่านได้ 0 อักขระหรือไม่ก็อ่านได้ VMIN อักขระแต่กรรมวิธีดังกล่าวก็มีความเสี่ยงกรณีมีการสูญหายของอักขระเพราะจะทำให้การอ่านถูกบล็อกไปจนกว่าจะมีอักขระมาเพิ่มจนครบจำนวน VMIN

VTIME ใช้ระบุเวลาที่รออ่านอักขระโดย 1 VTIME เท่ากับ 0.1 วินาทีและในกรณีที่เซต VTIME เป็น 0 การอ่านจะถูกบล็อกไปแบบไม่จำกัด แต่การบล็อกจะถูกยกเว้นถ้ามีการเซตตัวเลือก NDELAY บนแฟ้มผ่านฟังก์ชัน open() หรือ fcntl()

แต่อย่างไรก็ตามบทนิยาม VMIN และ VTIME ก็อาจแตกต่างจากที่ได้กล่าวข้างต้นโดยบางเอกสารก็ให้บทนิยาม VMIN และ VTIME ดังนี้

1. ถ้า VMIN > 0 และ VTIME = 0 การอ่านจะสิ้นสุดเมื่ออักขระที่อ่านได้มีจำนวนไม่ต่ำกว่า VMIN

2. ถ้า VMIN = 0 และ VTIME > 0 การอ่านจะสิ้นสุดเมื่ออ่านได้ 1 อักขระหรือเวลาที่รอเกิน 0.1*VTIME วินาทีซึ่งในกรณีหลังนี้จะถือว่าอ่านอักขระได้ 0 อักขระหรือก็คืออ่านอักขระไม่ได้เลย

3. ถ้า VMIN > 0 และ VTIME > 0 การรับจะสิ้นสุดเมื่ออักขระที่ได้รับมีจำนวนไม่ต่ำกว่า VMIN หรือระยะเวลาห่างระหว่างอักขระที่รอเกิน 0.1*VTIME วินาทีโดยการจับเวลาจะเริ่มนับหลังจากที่ได้อ่านอักขระแรกแล้วและการจับเวลาจะเริ่มนับใหม่หลังจากการอ่านอักขระล้าสุด

4. ถ้า VMIN = 0 และ VTIME = 0 การอ่านจะสิ้นสุดทันทีเพราะการอ่านแบบนี้จะอ่านอักขระเท่าที่มีอยู่ในบัฟเฟอร์

2.7.4 การโปรแกรมระดับสูงช่องทางข้อมูลอนุกรม

ในบทนี้จะกล่าวถึงการโปรแกรมระดับสูงช่องทางข้อมูลอนุกรมด้วยฟังก์ชัน ioctl() และ select()

2.7.4.1 ฟังก์ชัน ioctl()

เป็นฟังก์ชันภายใต้ยูนิกซ์ที่เพิ่มความสามารถในการโปรแกรมให้มากขึ้นกว่าการใช้ฟังก์ชัน tcgetattr() และ tcsetattr() ที่ได้แนะนำในบทที่ผ่านมา

```
#include <unistd.h>
#include <termios.h>
int ioctl(int fd, int request, ...);
```

ซึ่งฟังก์ชัน ioctl() ต้องการ 2 อาร์กิวเมนต์ดังต่อไปนี้

1. fd เพื่อระบุแฟ้มที่ต้องการจะโปรแกรมด้วย
2. request เพื่อระบุการกระทำที่ต้องการโดยใช้ค่าคงที่ดังต่อไปนี้

ตาราง 2.12 ตัวเลือกของ request

Request	Description	POSIX Function
TCGETS	Gets the current serial port settings.	tcgetattr
TCSETS	Sets the serial port settings immediately.	tcsetattr(fd, TCSANOW, &options)
TCSETSF	Sets the serial port settings after flushing the input and output buffers.	tcsetattr(fd, TCSAFLUSH, &options)
TCSETSW	Sets the serial port settings after allowing the input and output buffers to drain/empty.	tcsetattr(fd, TCSADRAIN, &options)
TCSBRK	Sends a break for the given time.	tcsendbreak, tcdrain
TCXONC	Controls software flow control.	tcflow
TCFLSH	Flushes the input and/or output queue.	tcflush
TIOCMGET	Returns the state of the "MODEM" bits.	None
TIOCMSET	Sets the state of the "MODEM" bits.	None
FIONREAD	Returns the number of bytes in the input buffer.	None

จะเห็นว่าฟังก์ชัน ioctl() สามารถทำงานได้หลากหลายตามตัวเลือกในตาราง 2.12 ซึ่งในที่นี้จะแนะนำการใช้งานตัวเลือกบางตัวเช่น FIONREAD

- FIONREAD ใช้อ่านจำนวนไบต์ในอินพุตบัฟเฟอร์ของช่องทางข้อมูลอนุกรมดังนี้

```
#include <unistd.h>
#include <termios.h>
int fd;
int bytes;
ioctl(fd, FIONREAD, &bytes);
```

โดยอาร์กิวเมนต์ bytes ใช้ระบุที่เก็บจำนวนไบต์ในอินพุตบัฟเฟอร์

2.7.4.2 **ฟังก์ชัน** select()

การโปรแกรมที่ต้องจัดการช่องทางข้อมูลหลายช่องพร้อมกันจะมีความสะดวกมากขึ้นหากสามารถตรวจสอบไบต์ข้อมูลในอินพุตบัฟเฟอร์ของทุกช่องได้ในคำสั่งเดียว ดังนั้นยูนิกซ์จึงจัดหาฟังก์ชัน select() ขึ้นมาเพื่อใช้ตรวจสอบอินพุต เอาต์พุต ข้อผิดพลาดที่เกิดกับช่องสัญญาณซึ่งสามารถตรวจได้พร้อมกันหลายช่องและตัวบอกแฟ้มนี้ก็สามารถชี้ไปได้ทั้งช่องทางข้อมูลอนุกรม แฟ้มทั่วไป อุปกรณ์อื่นใด ท่อ ซ็อกเก็ต ฯลฯ นอกจากนี้ฟังก์ชัน select() ยังกำหนดระยะหมดเวลารอที่จะตรวจสอบไบต์ได้อีกด้วย

```
int select(int max_fd, fd_set *input, fd_set *output, fd_set *error, struct timeval *timeout);
```

โดยต้องการ 5 อาร์กิวเมนต์ดังนี้

1. max_fd เพื่อระบุหมายเลขสูงสุดของตัวบอกในเซต input output error
2. input เพื่อระบุเซตตัวบอกอินพุต
3. output เพื่อระบุเซตตัวบอกเอาต์พุต
4. error เพื่อระบุเซตตัวบอกควมผิดพลาด

ซึ่งเซตของตัวบอกสามารถสร้างได้จาก 3 แมโครต่อไปนี้

```
FD_ZERO(&fd_set);
FD_SET(fd, &fd_set);
FD_CLR(fd, &fd_set);
```

5. timeout เพื่อระบุระยะหมดเวลารอด้วย timeout.tv_sec และ timeout.tv_usec

ฟังก์ชัน select() จะรีเทิร์นจำนวนของตัวบอกตามเงื่อนไขที่ระบุใน fd_set หรือรีเทิร์น -1 ในกรณีมีความผิดพลาด

2.8 **ซีอาร์ซี** (8)

2.8.1 **บทนำซีอาร์ซี**

การตรวจสอบด้วยส่วนซ้ำซ้อนแบบวนหรือซีอาร์ซี (CRC : Cyclic redundancy check) ถือเป็นฟังก์ชันแฮชชนิดหนึ่งที่ใช้สร้างผลรวมตรวจสอบบล็อกข้อมูลที่มีจำนวนบิตคงที่ค่าใด ๆ เช่น ใช้ตรวจสอบกลุ่มข้อมูลที่รับส่งผ่านเครือข่ายหรือใช้ตรวจสอบแฟ้มข้อมูลในคอมพิวเตอร์ โดยซีอาร์ซีจะถูกคำนวณและผนวกเข้ากับบล็อกข้อมูลก่อนที่จะส่งออกไปหรือเก็บในหน่วยเก็บและซีอาร์ซีนี้จะถูกตรวจสอบในภายหลังเพื่อใช้ยืนยันว่าข้อมูลภายในบล็อกไม่มีการเปลี่ยนแปลงเกิดขึ้นระหว่างการรับส่งหรือระหว่างเก็บในหน่วยเก็บ

ซีอาร์ซีเป็นที่นิยมนำมาใช้งานเพราะมีองค์ประกอบที่ไม่ซับซ้อนและสร้างได้ง่ายในทางฮาร์ดแวร์นอกจากนี้ยังสะดวกในการวิเคราะห์ทางคณิตศาสตร์และยังมีความสามารถในการตรวจหาความผิดพลาดของบล็อกข้อมูลจากสัญญาณรบกวนที่มักเกิดระหว่างช่องทางสื่อสาร

ซีอาร์ซีหรือผลรวมตรวจสอบคือเศษของการหารกระแสบิตข่าวสารด้วยกระแสบิตความยาว n บิตที่ได้กำหนดไว้โดยแต่ละบิตจะแทนสัมประสิทธิ์ของพหุนามและจะต้องผนวก 0 จำนวน n บิตเข้ากับกระแสบิตข่าวสารก่อนที่จะทำการหาร ซึ่งการหารเพื่อหาซีอาร์ซีนี้จะเป็นการหารแบบไม่มีการทดโดยจะใช้การออร์เฉพาะ (XOR : Exclusive-or) บนจีเอฟ 2 (GF2) หรือมอดุโล 2

XOR	0	1
0	0	1
1	1	0

ตัวอย่างการบวกแบบมอดุโล 2

$$(x^2 + x)+(x + 1) = x^2 + 2x + 1 = x^2 + 1$$

ตัวอย่างการคูณแบบมอดุโล 2

$$(x^2 + x)(x + 1) = x^3 + x^2 + x^2 + x = x^3 + 2x^2 + x = x^3 + x$$

ตัวอย่างการหารแบบมอดุโล 2

$$(x^3 + x^2 + x) / (x + 1) = (x^2 + 1) - (1 / (x + 1))$$

ซึ่งได้ผลลัพธ์จากการหารคือ $x^2 + 1$ และเหลือเศษจากการหารคือ -1

กระแสบิตความยาว n บิตที่กล่าวมาแล้วเมื่อนำมาเขียนเป็นสัมประสิทธิ์ของพหุนามจะเห็นได้ว่าค่า n ก็คือดีกรีของพหุนามและการผนวก 0 จำนวน n บิตเข้ากับกระแสบิตข่าวสารก่อนที่จะทำการหารก็คือการคูณกระแสบิตข่าวสารด้วย x^n นั้นเองส่วนพหุนามเศษเหลือที่เหลือจากการหารเมื่อนำสัมประสิทธิ์มาเขียนเป็นกระแสบิตความยาว n บิตก็จะได้ซีอาร์ซีที่ต้องการ ซึ่งถ้ามองในมุมของการหาค่าซีอาร์ซีแล้วมักจะเรียกพหุนามของกระแสบิตความยาว n บิตที่ได้กำหนดไว้อีกชื่อหนึ่งว่าพหุนามก่อกำเนิด

รูปแบบทั่วไป

$$M(x).x^n = Q(x).G(x) + R(x)$$

โดย M(x) แทนพหุนามของกระแสบิตข่าวสาร

Q(x) แทนพหุนามของผลลัพธ์จากการหาร

G(x) แทนพหุนามของกระแสบิตความยาว n บิตที่ได้กำหนดไว้

R(x) แทนพหุนามเศษเหลือจากการหาร

โดยทั่วไปเมื่อหาซีอาร์ซีที่ต้องการได้แล้วมักจะนำไปวางต่อกับบล็อกข้อมูลของซีอาร์ซีนั้น ๆ เพื่อใช้ในการตรวจสอบบล็อกข้อมูลในภายหลัง โดยการตรวจสอบทำได้โดยนำ G(x) ไปหาร

$M(x).x^n$ - $R(x)$ แล้วดูว่าได้เศษเท่ากับ 0 หรือไม่ ถ้าได้เศษจากการหารเป็น 0 กระบวนการตรวจสอบจะตั้งสมมติฐานว่าบล็อกข้อมูลนั้นถูกต้องซึ่งถ้ามองในมุมการตรวจสอบจะเห็นได้ว่า G(x) มีความสำคัญเสมือนเป็นกุญแจในการตรวจสอบจึงมีการเรียก G(x) อีกชื่อว่าพหุนามกุญแจ นอกจากนี้จะสังเกตได้ว่า $M(x).x^n$ - R(x) ก็คือตัวข้อมูลจริงหรือคำรหัสนั้นเอง

ซีอาร์ซีเป็นผลรวมตรวจสอบที่มาจากการคำนวณทางคณิตศาสตร์เพราะฉะนั้นจึงนำไปใช้กับมอดุโลอื่นได้เช่นกันตัวอย่างเช่นมอดุโล 10 มอดุโล 256 มอดุโล 65535 เป็นต้น

ซีอาร์ซีไม่เพียงแต่ใช้ตรวจหาความผิดพลาดได้เท่านั้นแต่ยังสามารถใช้เป็นส่วนหนึ่งในการแก้ไขข้อมูลที่ผิดพลาดให้กลับมาถูกต้องได้ในระดับหนึ่งโดยอาศัยพื้นฐานการคำนวณทางคณิตศาสตร์

2.8.2 **เอ็นเดียน**

การส่งข้อมูลแบบอนุกรมนั้นโดยทั่วไปจะให้บิตแรกที่ส่งไปเป็นบิตที่มีค่าสัมประสิทธิ์ของ x สูงที่สุดส่วน n บิตสุดท้ายที่ส่งก็คือซีอาร์ซีของบล็อกข้อมูลนั้น ๆ เริ่มต้นจากบิตที่มีสัมประสิทธิ์ x^{n-1} ไปจนถึงบิตสุดท้ายที่มีสัมประสิทธิ์ x^0

แต่ข้อมูลที่รับส่งในคอมพิวเตอร์มักรับส่งเป็นไบต์ผ่านบัสข้อมูลแบบขนานทำให้ลำดับของบิตในไบต์มีความสำคัญเวลานำมาคำนวณหาซีอาร์ซี ซึ่งจะต้องระบุว่าบิตไหนของไบต์เป็นบิตแรกที่จะถูกนำมากำหนดเป็นสัมประสิทธิ์ของ x ที่มีกำลังสูงสุดโดยเรียกการระบุนี้ว่า “เอ็นเดียน”

ตัวอย่างเอ็นเดียนตามมาตรฐานของอีเทอร์เน็ตและอาร์เอส-232 จะกำหนดใช้การส่งแบบ “ลิตเติ้ลเอ็นเดียน” หรือกำหนดให้บิตที่มีความสำคัญน้อยที่สุดในไบต์เป็นบิตแรก ดังนั้นการคำนวณซีอาร์ซีของการส่งแบบลิตเติ้ลเอ็นเดียนก็จะกำหนดให้บิตที่มีความสำคัญน้อยที่สุดในแต่ละไบต์เป็นสัมประสิทธิ์ของ x ที่มีกำลังสูงที่สุดแต่ในทางตรงข้ามแผ่นบันทึกและจานบันทึกแบบแข็งจะใช้การเขียนแบบ “บิ๊กเอ็นเดียน” หรือกำหนดให้บิตที่มีความสำคัญมากที่สุดในไบต์เป็นบิตแรก

การคำนวณซีอาร์ซีแบบลิตเติ้ลเอ็นเดียนนั้นดูเหมือนจะง่ายและสะดวกในการคำนวณทางซอฟต์แวร์มากกว่าแบบบิ๊กเอ็นเดียนดังนั้นโดยทั่วไปจะเห็นการคำนวณซีอาร์ซีแบบลิตเติ้ลเอ็นเดียนมากกว่าแต่ผู้เขียนโปรแกรมบางส่วนก็เห็นว่าบิ๊กเอ็นเดียนมีการเรียงลำดับของบิตที่สามารถดูหรือติดตามได้ง่ายกว่าแบบลิตเติ้ลเอ็นเดียนตัวอย่างเช่นซีอาร์ซีของเอกซ์โมเด็มหรือการคำนวณซีอาร์ซีในสมัยเริ่มแรกก็ใช้การคำนวณซีอาร์ซีแบบว่าบิ๊กเอ็นเดียน

2.8.2.1 การกำหนดเอ็นเดียน

สมมติว่าต้องการคำนวณ 8 บิตซีอาร์ซีของข้อมูลที่มีความยาว 8 บิตคืออักขระ 'W' ในแอสกีซึ่งเทียบได้กับ 87_{10} หรือ 57_{16} โดยใช้ CRC-8-ATM ที่มีพหุนาม $x^8 + x^2 + x + 1$ เป็นพหุนามก่อกำเนิดและจะกำหนดให้บิตแรกคือสัมประสิทธิ์ของ x ที่มีกำลังสูงที่สุดโดยจะเขียนให้อยู่ทางซ้ายมือ ซึ่งได้ผลออกมาเป็น "100000111"

ไบต์ข้อมูล W สามารถส่งได้ 2 แบบขึ้นกับการกำหนดเอ็นเดียนซึ่ง W สามารถเขียนในรูปบิตได้ดังนี้ 01010111 โดยบิตที่มีความสำคัญมากที่สุดจะอยู่ทางซ้ายส่วนบิตที่มีความสำคัญน้อยที่สุดจะอยู่ทางขวา ในกรณีบิ๊กเอ็นเดียนสามารถเขียน W ในรูปพหุนามได้เป็น $x^6 + x^4 + x^2 + x + 1$ หรือ 01010111 ส่วนกรณีลิตเติ้ลเอ็นเดียนสามารถเขียน W ในรูปพหุนามได้เป็น $x^7 + x^6 + x^5 + x^3 + x$ หรือ 11101010 ซึ่งทั้ง 2 กรณีจะให้ซีอาร์ซีที่แตกต่างกันและก่อนที่จะคำนวณซีอาร์ซีจะต้องคูณพหุนามข้อมูล M(x) ด้วย x^8 ซึ่งเทียบได้กับการเติม 0 จำนวน 8 ตัวต่อท้ายอักขระ 'W'

การคำนวณเพื่อหาเศษที่เหลือจากการหารจะใช้การคำนวณแบบมอดุโล 2 ดังที่ได้อธิบายไว้ในตอนต้นกล่าวคือไม่มีการทดและตัวเลขมีเพียง 0 และ 1 เท่านั้น

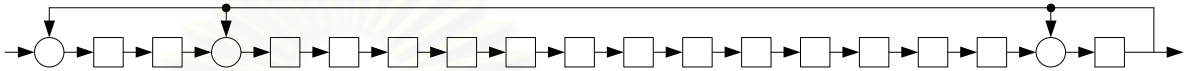
ตาราง 2.13 การคำนวณหาซีอาร์ซีจากเศษเหลือของการหารแบบมอดุโล 2

Big-endian example																
	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0
-	0	0	0	0	0	0	0	0	0							
=	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0
	-	1	0	0	0	0	0	1	1	1						
=	0	0	0	1	0	1	1	0	1	1	0	0	0	0	0	0
		-	0	0	0	0	0	0	0	0	0					
=	0	0	0	1	0	1	1	0	1	1	0	0	0	0	0	0
			-	1	0	0	0	0	0	1	1	1				
=	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0	0
				-	0	0	0	0	0	0	0	0	0			
=	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0	0
					-	1	0	0	0	0	0	1	1	1		
=	0	0	0	0	0	0	1	0	1	0	1	0	1	1	0	0
						-	1	0	0	0	0	0	1	1	1	
=	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0
							-	0	0	0	0	0	0	0	0	0
=	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0

Little-endian example																
	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0
-	1	0	0	0	0	0	1	1	1							
=	0	1	1	0	1	0	0	1	1	0	0	0	0	0	0	0
	-	1	0	0	0	0	0	1	1	1						
=	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0
		-	1	0	0	0	0	0	1	1	1					
=	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0
			-	0	0	0	0	0	0	0	0	0				
=	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0
				-	1	0	0	0	0	0	1	1	1			
=	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0
					-	0	0	0	0	0	0	0	0	0		
=	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0
						-	0	0	0	0	0	0	0	0	0	
=	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0

บิ๊กเอ็นเดียนจะได้เศษจากการหารที่เขียนในรูปพหุนามได้เป็น $x^7 + x^5 + x$ หรือเท่ากับ $A2_{16}$ (10100010) ส่วน ลิตเติ้ลเอ็นเดียนจะได้เศษจากการหารที่เขียนในรูปพหุนามได้เป็น $x^7 + x^4 + x^3$ ซึ่งสามารถเปลี่ยนเป็นเลขฐาน 2 ตามวิธีของลิตเติ้ลเอ็นเดียนได้เท่ากับ 00011001 หรือ 19_{16}

ตาราง 2.14 การหาซีอาร์ซี 16 ของ ABC แบบลิตเติ้ลเอ็นเดียนจากเรจิสเตอร์แบบเลื่อน



Little-endian "ABC"

<16 zeros add><msb-**C**-lsb><msb-**B**-lsb><msb-**A**-lsb>	Modulo 2		Modulo 2														Modulo 2			clock
0000000000000000**01000011010000100100000 1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
0000000000000000**01000011010000100100000 0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		1
0000000000000000**01000011010000100100000**	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		2
									14 clocks pass away											
0000000000000000**01000011**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	0		16
0000000000000000**0100001**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**		17
0000000000000000**010000**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**		18
0000000000000000**01000**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**1**		19
0000000000000000**0100**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**		20
0000000000000000**010**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**		21
0000000000000000**01**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**		22
0000000000000000**0**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**		23
0000000000000000	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**		24
000000000000000	0	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**		25
00000000000000	1	0	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**		26
0000000000000	1	1	1	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**		27
000000000000	1	1	0	1	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**		28
00000000000	1	1	0	0	1	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**		29
0000000000	1	1	0	0	0	1	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**		30
000000000	1	1	0	0	0	0	1	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**		31
00000000	0	1	1	0	0	0	0	1	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**		32
0000000	1	0	0	1	0	0	0	0	1	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**		33
000000	0	1	0	0	1	0	0	0	0	1	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**		34
00000	0	0	1	0	0	1	0	0	0	0	1	**1**	**1**	**1**	**1**	**0**	**0**	**0**		35
0000	0	0	0	1	0	0	1	0	0	0	0	1	**1**	**1**	**1**	**1**	**1**	**0**		36
000	1	0	1	0	1	0	0	1	0	0	0	0	1	**1**	**1**	**1**	**0**	**1**		37
00	0	1	0	1	0	1	0	0	1	0	0	0	0	1	**1**	**1**	**1**	**0**		38
0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	1	**1**	**0**	**1**		39
	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	**0**		40

|→ 0100 = 4H ←| |→ 0101 = 5H ←| |→ 0010 = 2H ←| |→ 0001 = 1H ←|

CRC16 of "**ABC**" is "**4521**H" or character "**E!**" in ASCII

ตาราง 2.15 การคำนวณหาซีอาร์ซี 16 ของ ABC แบบลิตเติ้ลเอ็นเดียนจากเศษเหลือของการหารแบบมอดุโล 2

	lsb			A				msb	lsb			B				msb	lsb			C				msb	**16 zeros added**															
	1	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1																							
=	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1																						
	=	0	1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1																					
		=	0	1	0	0	1	0	0	1	0	0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
			-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1																				
			=	0	1	0	1	0	0	1	0	0	0	0	0	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
				-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1																			
				=	0	1	1	0	0	1	0	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
					-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1																		
					=	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
									-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1														
									=	0	1	0	0	0	0	1	1	1	0	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
										-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1													
										=	0	1	0	0	0	1	1	1	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
											-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1												
											=	0	1	0	0	1	1	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
												-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1											
												=	0	1	0	1	1	1	0	0	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
													-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1										
													=	0	1	1	1	1	0	0	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
														-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1									
														=	0	0	1	1	0	0	1	1	1	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
																-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1							
																=	0	0	0	0	1	1	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
																				-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1			
																				=	0	0	1	1	1	0	0	0	0	1	0	0	1	0	1	0	1	0	0	0
																						-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	
																						=	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0
																									or 0100 0101 0010 0001 = 4521H															

Remark minus sign '-' equivalent to modulo 2

CRC16 of **ABC** is **0x4521**

2.9 **วอตช์ดอก** (2)

2.9.1 **บทนำวอตช์ดอก**

ตัวจับเวลาวอตช์ดอก (Watchdog Timer) เป็นวงจรฮาร์ดแวร์ที่สามารถตั้งใหม่ระบบคอมพิวเตอร์ในกรณีที่ซอฟต์แวร์เกิดความผิดพร่อง โดยทั่วไปในขณะที่ปริภูมิผู้ใช้ยังคงทำงานได้ตามปกติ ดีมอนที่อยู่ในปริภูมิผู้ใช้จะบอกโปรแกรมขับวอตช์ดอกในเคอร์เนลผ่านแฟ้มอุปกรณ์พิเศษ /dev/watchdog ด้วยระยะห่างระหว่างการบอกเท่าๆ กัน เมื่อโปรแกรมขับวอตช์ดอกได้รับการบอกมาจากดีมอนทางโปรแกรมขับก็จะบอกไปยังวอตช์ดอกที่เป็นวงจรฮาร์ดแวร์เพื่อแสดงว่าทุกอย่างยังคงทำงานได้เป็นปกติดี แต่ถ้าปริภูมิผู้ใช้เกิดความขัดข้องบางประการเช่น แรมเกิดความผิดพลาด มีจุดบกพร่องในเคอร์เนล ฯลฯ ทำให้ดีมอนที่อยู่ปริภูมิผู้ใช้ไม่สามารถบอกโปรแกรมขับได้ตามปกติเป็นผลให้โปรแกรมขับไม่ได้บอกต่อไปยังวอตช์ดอกที่เป็นวงจรฮาร์ดแวร์และถ้าหากระยะห่างระหว่างการบอกมากกว่าที่กำหนดหรือถึงจุดหมดเวลารอทางวอตช์ดอกที่เป็นวงจรฮาร์ดแวร์ก็จะทำการตั้งใหม่ระบบทันที

ลินุกซ์วอตช์ดอกเอพีไอมีการพัฒนาโปรแกรมขับที่แตกต่างกันออกไปเพราะว่าไม่มีจุดอ้างอิงที่แน่นอนสำหรับการพัฒนาโปรแกรมขับทำให้บางครั้งเกิดเข้ากันไม่ได้หรือใช้แทนกันไม่ได้ในบางส่วน จึงมีผู้พยายามจัดทำเอกสารเพื่อใช้เป็นจุดอ้างอิงสำหรับการพัฒนาโปรแกรมขับในอนาคต

2.9.2 **เอพีไออย่างง่าย**

วอตช์ดอกควรจะอยู่ในสภาวะทำงานเมื่อเปิดแฟ้ม /dev/watchdog และจะทำการตั้งใหม่ระบบถ้าวอตช์ดอกไม่ได้รับการปิง (ping) ภายในระยะเวลาที่แน่นอนค่าหนึ่งที่เรียกว่าหมดเวลารอ (timeout) หรือขอบ (margin) วิธีง่ายสุดที่จะปิงวอตช์ดอกคือการเขียนข้อมูลบางอย่างไปยังวอตช์ดอก

โปรแกรมขับระดับสูงอาจตรวจสอบแม่ข่ายเอชทีทีพีว่ายังตอบสนองได้ปกติหรือไม่ก่อนการเขียนเพื่อจะปิงวอตช์ดอก แต่เมื่ออุปกรณ์ถูกปิดก็จะปิดทางวอตช์ดอกไปด้วยซึ่งถือเป็นแนวคิดที่ไม่ค่อยจะดีนักเพราะหากวอตช์ดอกดีมอนมีจุดบกพร่องจนเกิดล้มเหลวในการทำงานก็จะทำให้ระบบไม่มีทางตั้งใหม่โดยวอตช์ดอกได้ ด้วยเหตุนี้ทำให้โปรแกรมขับบางโปรแกรมจะอนุญาตให้ปิดทางวอตช์ดอกได้ก็ต่อเมื่อต้องปิดเครื่องเท่านั้น โดยเลือกตัวบ่งชี้ CONFIG_WATCHDOG_NOWAYOUT เป็น Y ขณะแปลโปรแกรมเคอร์เนลซึ่งทำให้การปิดทางวอตช์ดอกไม่สามารถทำได้หลังจากวอตช์ดอกเริ่มต้นการทำงานไปแล้วและแม้ว่าวอตช์ดอกดีมอนจะล้มเหลวระบบก็จะถูกตั้งใหม่ทันทีหลังจากหมดเวลารอ

โปรแกรมขับบางโปรแกรมจะไม่มีการปิดทางวอตช์ดอกไปจนกว่าจะมีการส่งอักขระ V ไปให้ /dev/watchdog จึงจะสามารถปิดทางวอตช์ดอกได้ ถ้าหากดีมอนในปริภูมิผู้ใช้ถูกปิดโดยไม่ได้ส่ง

อักขระ V ทางโปรแกรมขับจะสมมติว่าดีมอนไม่ทำงานแล้วและจะหยุดการปิงไปยังวอตช์ดอกอันจะทำให้วอตช์ดอกสั่งการตั้งใหม่ระบบ

2.9.3 ไอโอซีทีแอลเอพีไอ (The ioctl API)

โปรแกรมขับที่เข้ากันได้ทุกโปรแกรมจะรองรับไอโอซีทีแอลเอพีไอ ซึ่งการปิงวอตช์ดอกก็จะใช้ไอโอซีทีแอลนี้เองโดยโปรแกรมขับจะต้องรองรับโปรแกรมต่อประสานไอโอซีทีแอลที่มีตัวบ่งชี้ KEEPALIVE เพื่อเขียนไปยังอุปกรณ์วอตช์ดอกโดยมีลักษณะโปรแกรมดังนี้

```
while (1) {
        ioctl(fd, WDIOC_KEEPALIVE, 0);
        sleep(10);
}
```

2.9.4 การเซตและการรับระยะหมดเวลารอ

โปรแกรมขับบางโปรแกรมอนุญาตให้ดัดแปรระยะหมดเวลารอของวอตช์ดอกด้วย SETTIMEOUT ioctl ผ่านการเซตตัวบ่งชี้ WDIOF_SETTIMEOUT ซึ่งอาร์กิวเมนต์ที่เป็นจำนวนเต็มจะแสดงระยะหมดเวลารอโดยมีหน่วยเป็นวินาทีและทางโปรแกรมขับจะรีเทิร์นระยะหมดเวลารอจริงที่ใช้ในตัวแปรดังกล่าว แต่ระยะหมดเวลารออาจแตกต่างไปจากที่ร้องขอไปอันเนื่องมาจากข้อจำกัดทางฮาร์ดแวร์

```
int timeout = 45;
ioctl(fd, WDIOC_SETTIMEOUT, &timeout);
printf("The timeout was set to %d seconds\n", timeout);
```

ตัวอย่างข้างต้นอาจพิมพ์ "The timeout was set to 60 seconds" ถ้าหากอุปกรณ์ที่ใช้มีระยะหมดเวลารอในหน่วยนาทีและตั้งแต่เคอร์เนล 2.4.18 เป็นต้นมาการซักถามระยะหมดเวลารอสามารถทำได้ผ่าน GETTIMEOUT ioctl ดังนี้

```
ioctl(fd, WDIOC_GETTIMEOUT, &timeout);
printf("The timeout was is %d seconds\n", timeout);
```

บทที่ 3

การทดสอบ

3.1 ภาพรวมการทดสอบ

เป็นการทดสอบประสิทธิภาพของระบบฝังตัวที่จะนำมาใช้เป็นตัวควบคุมการสื่อสาร โดยมีความต้องการเบื้องต้นคือระบบสามารถรองรับการสื่อสารผ่านช่องทางข้อมูลอนุกรมได้พร้อมกันทั้ง 8 ช่องที่อัตราบอด 9600 บิตต่อวินาทีโดยมีปริมาณงาน 38400 บิตต่อวินาที ซึ่งในการทดสอบได้สมมติกรอบความยาว 70 ไบต์ขึ้นมาใช้ในการรับส่งด้วยวิธีการหยั่งสัญญาณและวิธีรับส่งสัญญาณโดยไม่มีการควบคุมสายงานทั้งทางซอฟต์แวร์และฮาร์ดแวร์ แต่มีการจับเวลาเพื่อใช้วัดประสิทธิภาพ ซึ่งแต่เดิมได้ทดสอบกับพีซี/104 รุ่น PCM-3341 + PCM-3643 ที่มีสถาปัตยกรรมแบบ x86 โดยใช้ลินุกซ์ (เคอร์เนล 2.6) ที่ผู้ทดสอบประกอบขึ้นเองเป็นระบบปฏิบัติการแต่เกิดปัญหาในการทดสอบคือไออาร์คิวที่ใช้ได้มีจำกัดทำให้ต้องใช้ไออาร์คิวร่วม แต่ว่าช่องทางข้อมูลอนุกรมบนพีซี/104 อยู่บนบัสแบบไอเอสเอซึ่งใช้สัญญาณขอขัดจังหวะแบบขอบ (edge interrupt) การจะใช้ไออาร์คิวร่วมได้ต้องมีโปรแกรมขับที่เหมาะสมซึ่งทางผู้ทดสอบไม่สามารถหาโปรแกรมขับดังกล่าวที่ใช้กับลินุกซ์ได้ เป็นผลให้ช่องทางข้อมูลอนุกรมทั้ง 8 ช่องของพีซี/104 ทำงานพร้อมกันไม่ได้ จึงต้องเปลี่ยนมาทดสอบกับอุปกรณ์ตัวใหม่คือ RISC base ที่มีสถาปัตยกรรมแบบ ARM รุ่น UC-7408 ที่มีลินุกซ์ (เคอร์เนล 2.4) ติดตั้งมาพร้อมแล้วและช่องทางข้อมูลอนุกรมทั้ง 8 ก็อยู่บนบัสแบบพีซีไอทำให้ปัญหาด้านไออาร์คิวหมดไปเพราะบัสแบบพีซีไอใช้สัญญาณขอขัดจังหวะแบบระดับ (level interrupt) ที่ใช้ไออาร์คิวร่วมได้แบบไม่เป็นปัญหาและผลของการทดสอบกับ UC-7408 ก็ปรากฏว่าช่องทางข้อมูลอนุกรมทั้ง 8 ช่องทำงานพร้อมกันได้ โดยที่อัตราบอด 9600 บิตต่อวินาทีที่ภาวะงานปกติด้วยวิธีรับส่งสัญญาณวัด excess time สูงสุดได้ 7.3 มิลลิวินาทีและได้ปริมาณงาน 70440 บิตต่อวินาที ส่วนขีดจำกัดของระบบคือที่สภาวะภาระงานสูงสุดระบบทำงานได้ถึงอัตราบอด 9600 บิตต่อวินาทีในกรรมวิธีการหยั่งสัญญาณและ 115200 บิตต่อวินาทีในกรรมวิธีรับส่งสัญญาณก่อนจะล้มเหลวด้วย Segmentation fault

3.2 วิธีทดสอบ

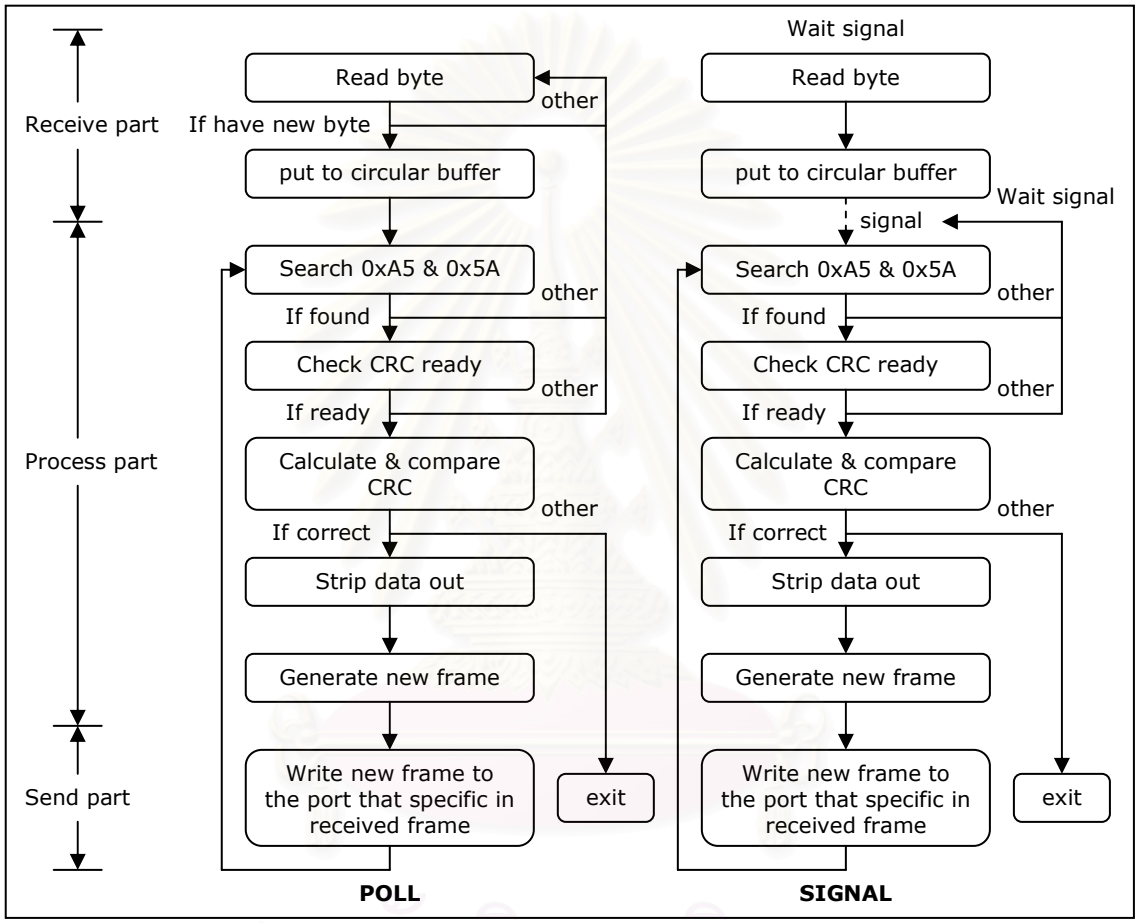
3.2.1 กรอบความที่ใช้ทดสอบ

ตาราง 3.1 กรอบความที่ใช้ทดสอบรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรม

Content	Objective	Length (bytes)
0xA5	Frame synchronize	1
0x5A	Frame synchronize	1
Destination	Specific destination of data	1
Data length	Specific length of data	1
Data	Contain real data	0 to 255
CRC16	Verify error	1
CRC16	Verify error	1

โดยซีอาร์ซี 16 จะเริ่มคำนวณจากไบต์แรกของกรอบความคือ A5 ไปจนถึงไบต์สุดท้ายของ Data ด้วยพหุนามก่อกำเนิด $X^{16}+X^{15}+X^{2}+X+1$ แบบลิตเติ้ลเอ็นเดียน ซึ่งในการทดสอบครั้งนี้จะแทน Data ที่มีความยาว 64 ไบต์ด้วย o--------oo--------oo--------oo--------oo--------oo--------oo--o ซึ่งแต่ละไบต์จะเป็นแบบ 8n1 คือมีบิตเริ่ม 1 บิต บิตอักขระ 8 บิต ไม่มีบิตภาวะคู่หรือคี่ และบิตหยุด 1 บิต ทำให้แต่ละไบต์จะประกอบไปด้วยบิตทั้งหมด 10 บิต

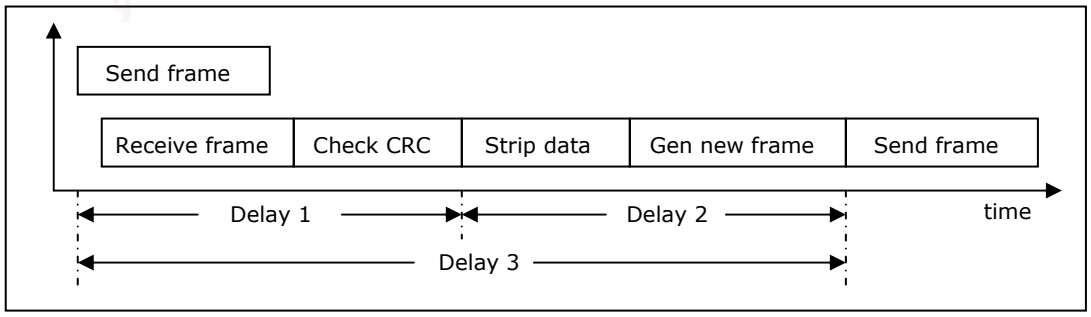
3.2.2 กรรมวิธีการหยั่งสัญญาณและรับส่งสัญญาณ



รูป 3.1 ขั้นตอนระหว่างการรับส่งกรอบความผ่านช่องทางข้อมูลอนุกรม

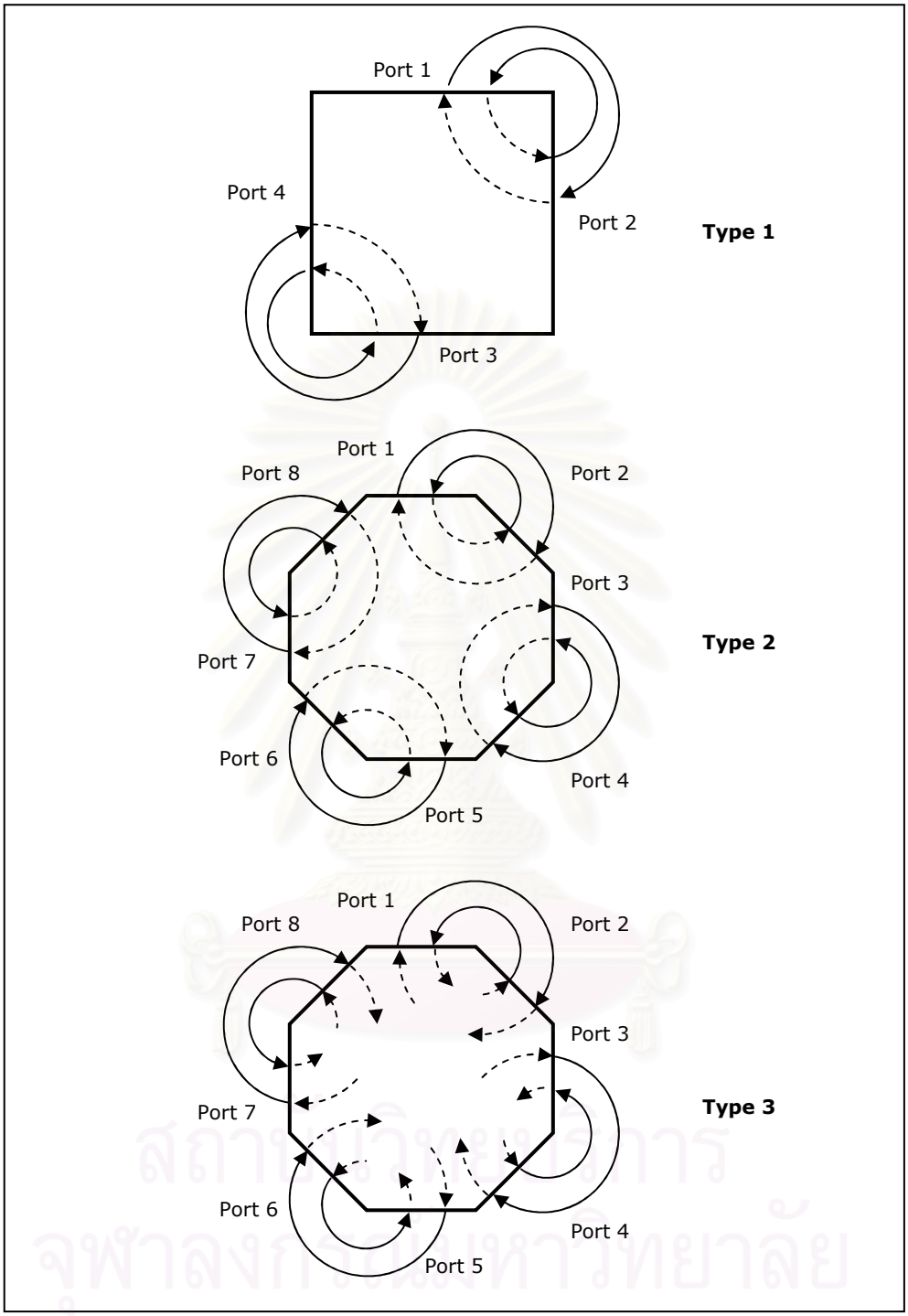
3.2.3 การจับเวลา

การจับเวลาในการทดสอบจะใช้ฟังก์ชัน gettimeofday() โดยมีจุดของการจับเวลาดังรูป 3.2



รูป 3.2 ตำแหน่งการจับเวลา

3.2.4 รูปแบบการทดสอบ



รูป 3.3 รูปแบบการทดสอบรับส่งผ่านช่องทางข้อมูลอนุกรม

1. แบบที่ 1 เริ่มต้นโดยส่งกรอบความออกทางช่องทางข้อมูลอนุกรมที่ 1-4 ช่องทางละ 1 กรอบความและให้กรอบความดังกล่าววิ่งวนอยู่ภายในแต่ละวงโดยมีขั้นตอนรับส่งตามรูป 3.1

2. แบบที่ 2 เริ่มต้นโดยส่งกรอบความออกทางช่องทางข้อมูลอนุกรมที่ 1-8 ช่องทางละ 1 กรอบความและให้กรอบความดังกล่าววิ่งวนอยู่ภายในแต่ละวงโดยมีขั้นตอนรับส่งตามรูป 3.1

3. แบบที่ 3 ส่งกรอบความออกทางช่องทางข้อมูลอนุกรมที่ 1-8 ตลอดเวลาที่สามารถส่งได้ ส่วนฝ่ายรับจะทำงานตามขั้นตอนดังรูป 3.1 ต่างกันเพียงจะไม่มีการส่งกรอบความใหม่ออกไปตามช่องทางข้อมูลอนุกรม

การทดสอบบน UC-7408 มีการใช้งานวอตช์ดอกโดยเซตให้ระยะหมดเวลารออยู่ที่ 2 วินาทีแต่กำหนดให้โปรแกรมส่งสัญญาณให้วอตช์ดอกทุก ๆ 1 วินาที ส่วนการทดสอบบน PCM-3341 + PCM-3643 ไม่มีการใช้งานวอตช์ดอก

3.3 เครื่องมือที่ใช้ในการทดสอบ

1. PC/104 รุ่น PCM-3341 + PCM-3643 ของ Advantech
2. RISC base embedded computer รุ่น UC-7408 ของ MOXA

3.4 การเก็บรวบรวมข้อมูล

ข้อมูลที่เก็บจะเป็นข้อมูลทางด้านเวลาเพื่อนำมาศึกษาและเปรียบเทียบโดยจะแยกเป็น 2 ส่วน คือข้อมูลที่ทดสอบบน PCM-3341 + PCM-3643 กับข้อมูลที่ทดสอบบน UC-7408

3.4.1 ข้อมูลที่ทดสอบบน PC/104 รุ่น PCM-3341 + PCM-3643

ตาราง 3.2 ผลทดสอบแบบที่ 1 โดยวิธีรับส่งสัญญาณที่การแบ่งเวลา 100 มิลลิวินาทีกับช่องทาง 1-4

Buad	Delay 1 (second)			Delay 2 (second)		
	Max	Mean	Min	Max	Mean	Min
50	13.989441	13.983359	13.977861	0.003803	0.002075	0.001715
75	9.331891	9.323105	9.320422	0.002712	0.002055	0.001727
110	6.363725	6.355993	6.353732	0.002677	0.002055	0.001698
134	5.227058	5.221670	5.219623	0.003690	0.002095	0.001721
150	4.671186	4.665630	4.661593	0.003109	0.002115	0.001777
200	3.507028	3.499971	3.497472	0.005062	0.002152	0.001730
300	2.346619	2.336062	2.333000	0.003040	0.002158	0.001868
600	1.214460	1.177006	1.169935	0.005661	0.002806	0.001833
1200	0.607038	0.592473	0.586056	0.003992	0.002905	0.001822
2400	0.323322	0.314506	0.307462	0.004373	0.002421	0.001794
4800	0.208021	0.164592	0.152927	0.006957	0.002924	0.001754
9600	0.098864	0.084975	0.080484	0.019861	0.003849	0.001551
19200	0.082295	0.053170	0.040620	0.022614	0.003873	0.001912
38400	0.056427	0.030455	0.023074	0.016002	0.003896	0.001886
57600	0.047470	0.024354	0.017097	0.011504	0.003899	0.001574
115200	0.019500	0.015178	0.011843	0.003485	0.002206	0.001805

ตาราง 3.3 ผลทดสอบแบบที่ 1 โดยวิธีรับส่งสัญญาณที่การแบ่งเวลา 100 มิลลิวินาทีกับช่องทาง 5-8

Buad	Delay 1 (second)			Delay 2 (second)		
	Max	Mean	Min	Max	Mean	Min
50	14.025720	14.018224	14.007160	0.002675	0.002089	0.001824
75	9.356252	9.345245	9.341283	0.003246	0.002193	0.001817
110	6.383477	6.370546	6.368024	0.003752	0.002190	0.001738
134	5.242785	5.235122	5.230737	0.004919	0.002391	0.001690
150	4.685803	4.677395	4.671509	0.004613	0.002294	0.001766
200	3.535286	3.509506	3.505669	0.003487	0.002206	0.001801
300	2.354937	2.341329	2.338072	0.005160	0.002410	0.001750
600	1.445229	1.202227	1.172155	0.022023	0.004035	0.001656
1200	Unknown interrupt or fault at EIP ...					

ตาราง 3.4 ผลทดสอบแบบที่ 1 โดยวิธีการหยั่งสัญญาณที่การแบ่งเวลา 100 มิลลิวินาทีกับช่องทาง 1-4

Buad	Delay 1 (second)			Delay 2 (second)		
	Max	Mean	Min	Max	Mean	Min
50	14.258319	14.012543	13.973990	0.002466	0.002111	0.001583
75	9.602915	9.563236	9.318005	0.002471	0.002106	0.001633
110	6.578726	6.403022	6.351667	0.003251	0.002207	0.001613
134	5.517676	5.269621	5.217701	0.002803	0.002058	0.001543
150	4.879931	4.783839	4.660415	0.003924	0.002258	0.002009
200	3.796237	3.598474	3.498423	0.003153	0.002111	0.001581
300	2.617724	2.401007	2.331192	0.005425	0.002244	0.001877
600	1.368148	1.210106	1.166854	0.003467	0.002335	0.001825
1200	0.818601	0.773603	0.585042	0.003693	0.002205	0.001579
2400	0.607610	0.395595	0.293600	0.003171	0.002210	0.001623
4800	0.529919	0.367768	0.155989	0.003768	0.002522	0.001953
9600	0.402399	0.185605	0.078926	0.004085	0.002430	0.001621
19200	0.329905	0.125006	0.039713	0.002947	0.002242	0.001594
38400	0.321674	0.066971	0.021199	0.107196	0.005107	0.001534
57600	0.309737	0.055095	0.014228	0.004040	0.002123	0.001611
115200	0.234044	0.030564	0.008684	0.122703	0.005630	0.001786

ตาราง 3.5 ผลทดสอบแบบที่ 1 โดยวิธีการหยั่งสัญญาณที่การแบ่งเวลา 100 มิลลิวินาทีกับช่องทาง 5-8

Buad	Delay 1			Delay 2		
	Max	Mean	Min	Max	Mean	Min
50	14.319792	14.059715	14.005904	0.002828	0.002051	0.001781
75	9.605225	8.570496	9.340122	0.003248	0.002193	0.001855
110	6.608631	6.404124	6.367096	0.003052	0.002199	0.001654
134	5.527881	5.292022	5.228325	0.302130	0.010463	0.001601
150	4.908258	4.786809	4.669025	0.002733	0.002148	0.001617
200	3.769323	3.595949	3.503263	0.002798	0.002229	0.001666
300	2.564003	2.398211	2.337843	0.003904	0.002288	0.001630
600	Unknown interrupt or fault at EIP ...					

ตาราง 3.6 ผลทดสอบแบบที่ 1 โดยวิธีรับส่งสัญญาณที่การแบ่งเวลา 10 มิลลิวินาทีกับช่องทาง 1-4

Buad	Delay 1 (second)			Delay 2 (second)		
	Max	Mean	Min	Max	Mean	Min
50	13.989224	13.983055	13.977448	0.004909	0.002188	0.001708
75	9.330316	9.323060	9.319862	0.004602	0.002172	0.001756
110	6.361897	6.355690	6.353539	0.004413	0.002166	0.001834
134	5.227060	5.222100	5.220073	0.003576	0.002132	0.001797
150	4.686473	4.666435	4.662508	0.009195	0.002539	0.001718
200	3.507512	3.499903	3.497334	0.003492	0.002171	0.001800
300	2.348376	2.336014	2.333390	0.012920	0.002612	0.001683
600	1.202292	1.175018	1.169138	0.022791	0.005602	0.001699
1200	0.609773	0.593996	0.585959	0.004126	0.002850	0.002074
2400	0.331031	0.314177	0.305940	0.004714	0.002325	0.001766
4800	0.185335	0.164780	0.154314	0.055843	0.007845	0.001726
9600	0.123166	0.090848	0.078491	0.039060	0.007922	0.001792
19200	0.091875	0.060062	0.043027	0.012656	0.004046	0.001732
38400	0.163074	0.040506	0.022982	0.058979	0.007760	0.001655
57600	0.165988	0.032512	0.016350	0.068858	0.010403	0.001689
115200	0.060606	0.021889	0.009421	0.085663	0.008621	0.001604

ตาราง 3.7 ผลทดสอบแบบที่ 1 โดยวิธีรับส่งสัญญาณที่การแบ่งเวลา 10 มิลลิวินาทีกับช่องทาง 5-8

Buad	Delay 1			Delay 2		
	Max	Mean	Min	Max	Mean	Min
50	14.025518	14.017586	14.008426	0.005497	0.003074	0.001745
75	9.356460	9.345678	9.340618	0.005957	0.002905	0.001865
110	6.384035	6.374346	6.367364	0.012266	0.003054	0.001849
134	5.245901	5.236629	5.230653	0.014021	0.003216	0.001836
150	4.687990	4.677657	4.671533	0.009710	0.002502	0.001743
200	3.519251	3.508579	3.505899	0.016200	0.003085	0.001622
300	2.352102	2.341261	2.338482	0.003966	0.002348	0.001670
600	Unknown interrupt or fault at EIP ...					

ตาราง 3.8 ผลทดสอบแบบที่ 1 โดยวิธีการหยั่งสัญญาณที่การแบ่งเวลา 10 มิลลิวินาทีกับช่องทาง 1-4

Buad	Delay 1 (second)			Delay 2 (second)		
	Max	Mean	Min	Max	Mean	Min
50	14.007317	14.001810	13.983491	0.003982	0.002236	0.001712
75	9.354813	9.324303	9.317993	0.032098	0.002923	0.001775
110	6.779062	6.372741	6.352409	0.432271	0.021645	0.001661
134	5.242359	5.238734	5.219135	0.003287	0.002187	0.001587
150	4.681274	4.678086	4.660001	0.002713	0.002159	0.001703
200	3.522052	3.517706	3.498297	0.003197	0.002141	0.001602
300	2.753175	2.364240	2.330773	0.432183	0.021686	0.001872
600	1.589018	1.188484	1.166681	0.431990	0.027274	0.001808
1200	1.014442	0.606843	0.584224	0.435204	0.021703	0.001684
2400	0.334833	0.317752	0.310216	0.002930	0.002246	0.001686
4800	0.183189	0.158480	0.156120	0.003468	0.002238	0.001599
9600	0.401398	0.111616	0.075299	0.014591	0.003867	0.001996
19200	0.317972	0.074432	0.039500	0.050494	0.006053	0.001602
38400	0.367430	0.046131	0.020255	0.113462	0.009263	0.001752
57600	0.154442	0.031693	0.014613	0.140411	0.013064	0.001873
115200	0.127746	0.026962	0.007832	0.127292	0.009204	0.001659

ตาราง 3.9 ผลทดสอบแบบที่ 1 โดยวิธีการหยั่งสัญญาณที่การแบ่งเวลา 10 มิลลิวินาทีกับช่องทาง 5-8

Buad	Delay 1			Delay 2		
	Max	Mean	Min	Max	Mean	Min
50	14.295239	14.036891	14.004359	0.431262	0.032707	0.001668
75	9.364490	9.359720	9.341184	0.002714	0.002143	0.001721
110	6.640012	6.387258	6.365128	0.281534	0.013588	0.001688
134	5.258461	5.239953	5.229076	0.003003	0.002302	0.001739
150	4.699862	4.680350	4.669552	0.003242	0.002267	0.001633
200	3.531992	3.518116	3.505128	0.003209	0.002193	0.001742
300	2.361026	2.357304	2.338196	0.002701	0.002104	0.001630
600	1.200558	1.195796	1.169174	0.032418	0.003380	0.001630
1200	Unknown interrupt or fault at EIP ...					

3.4.2 ข้อมูลที่ทดสอบบน RISC base embedded computer รุ่น UC-7408

เนื่องจากข้อมูลทางเวลาที่ทดสอบบน UC-7408 นี้มีจำนวนมากและไม่เหมาะที่จะนำมาแสดงในรูปข้อมูลดิบผู้ทดสอบจึงได้แปรข้อมูลไปเป็นแผนภูมิซึ่งจะได้แสดงในหัวข้อถัดไป

3.5 การวิเคราะห์ข้อมูล

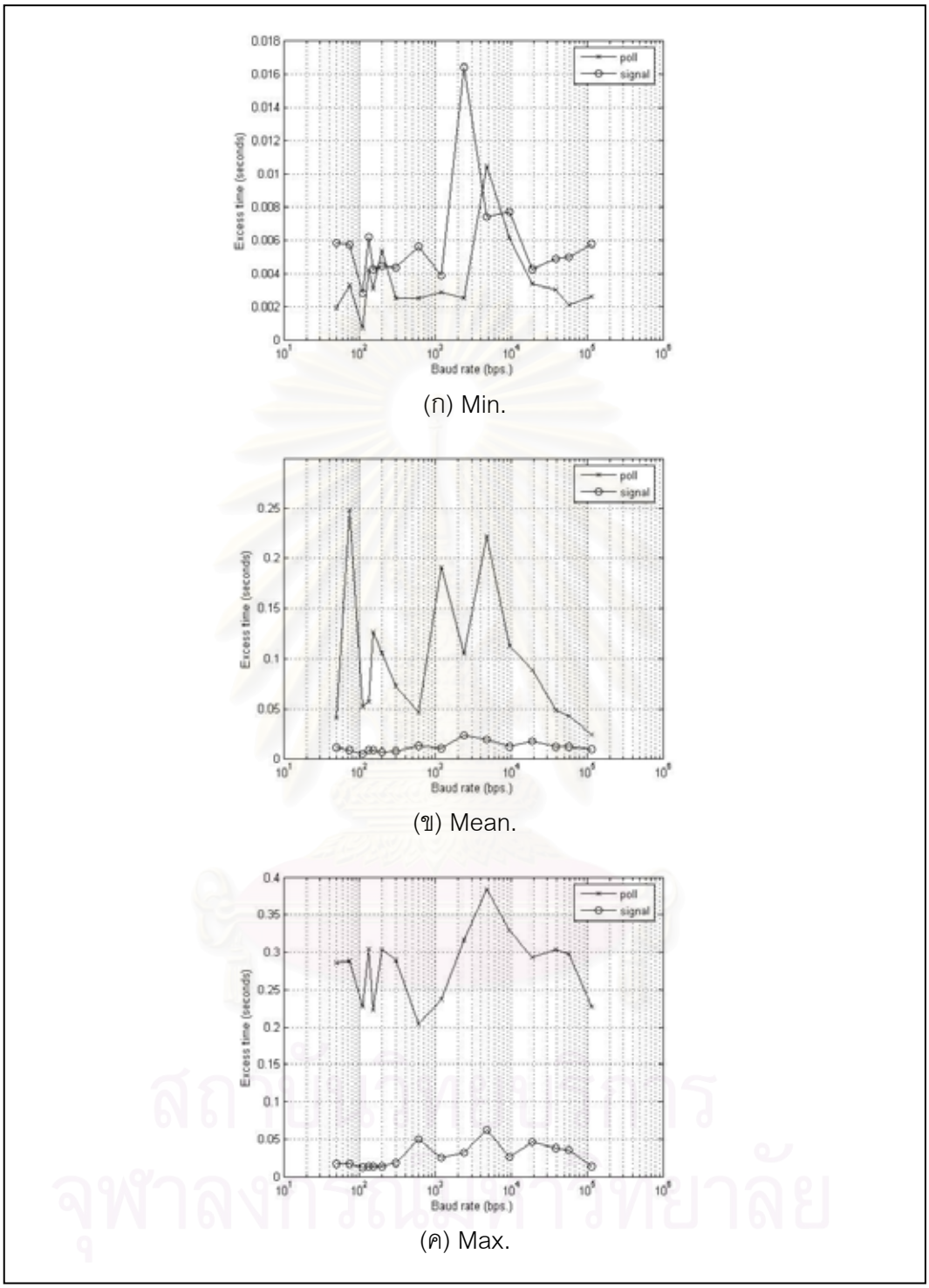
เพื่อความสะดวกในการวิเคราะห์ข้อมูลจึงได้แปรข้อมูลดิบไปเป็นแผนภูมิโดยเวลาที่แสดงจะลบเวลาที่ใช้ไปในการส่งข้อมูลผ่านช่องทางตามความเร็วของการส่งดังนี้

1. ใน PCM-3341 + PCM-3643 ให้ลบด้วย 700/(1.002xอัตราบอด) โดยใช้หน่วยมิลลิวินาที ซึ่งสาเหตุที่อัตราบอดต้องคูณด้วย 1.002 เป็นเพราะต้องการให้เวลาหลังการลบมีค่าเป็นบวกทุกค่า โดยคิดว่าช่องทางข้อมูลอนุกรมรับส่งได้เร็วกว่าที่ตั้งไว้ประมาณ 0.2 %

2. ใน UC-7408 ให้ลบด้วย 753/อัตราบอด โดยใช้หน่วยมิลลิวินาที ซึ่งตัวเลข 753 มาจากการจับเวลารับส่ง 1 กรอบความแล้วนำมาวาดกราฟหาจุดตัดที่อัตราบอดเท่ากับ 0 ซึ่งได้ค่าประมาณ 753 บิต

และให้เรียกระยะเวลาหลังการลบว่า excess time

3.5.1 PC/104 **รุ่น** PCM-3341 + PCM-3643

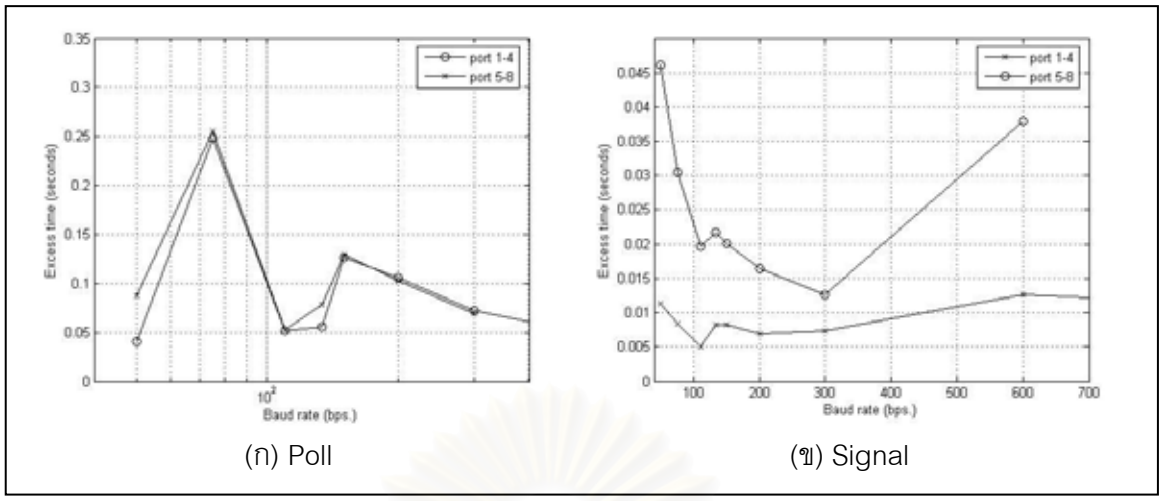


(ก) Min.

(ข) Mean.

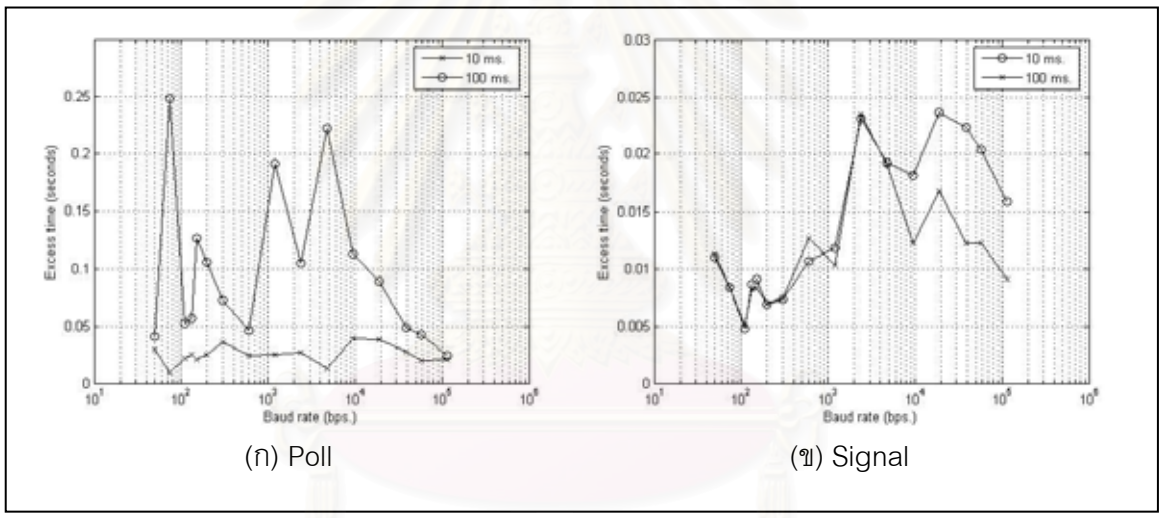
(ค) Max.

รูป 3.4 เปรียบเทียบ excess time ของ delay1 ของการทดสอบแบบที่ 1 ที่การแบ่งเวลา 100 มิลลิวินาทีผ่านช่องทาง 1-4 ระหว่างวิธีการหยั่งสัญญาณและวิธีการรับส่งสัญญาณ



(ก) Poll (ข) Signal

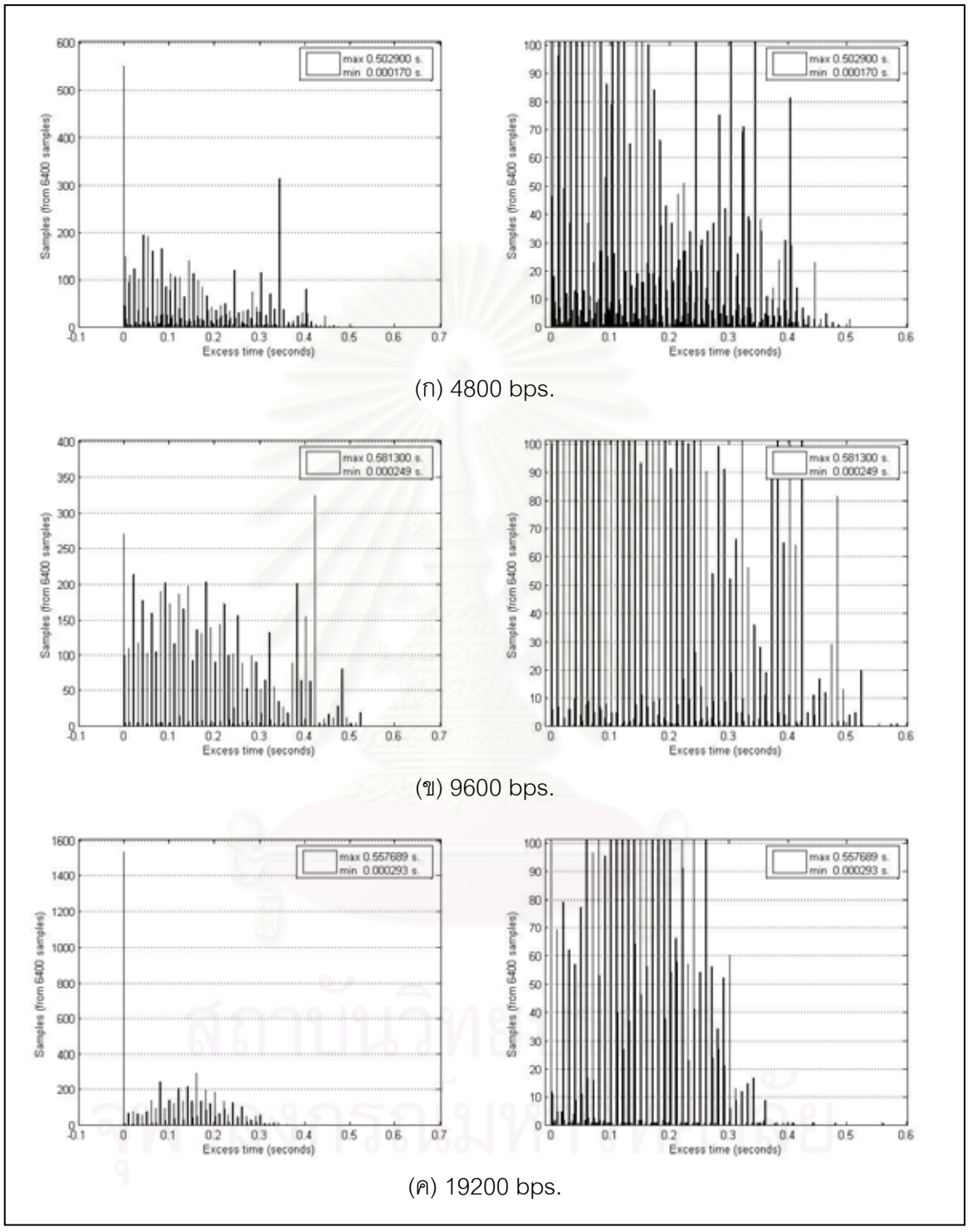
รูป 3.5 เปรียบเทียบ excess time ของค่าเฉลี่ย delay1 ของการทดสอบแบบที่ 1 ที่การแบ่งเวลา 100 มิลลิวินาทีระหว่างช่องทาง 1-4 และ 5-8



(ก) Poll (ข) Signal

รูป 3.6 เปรียบเทียบ excess time ของค่าเฉลี่ย delay1 ของการทดสอบแบบที่ 1 ผ่านช่องทาง 1-4 ระหว่างการแบ่งเวลา 10 และ 100 มิลลิวินาที

3.5.2 RISC base embedded computer รุ่น UC-7408

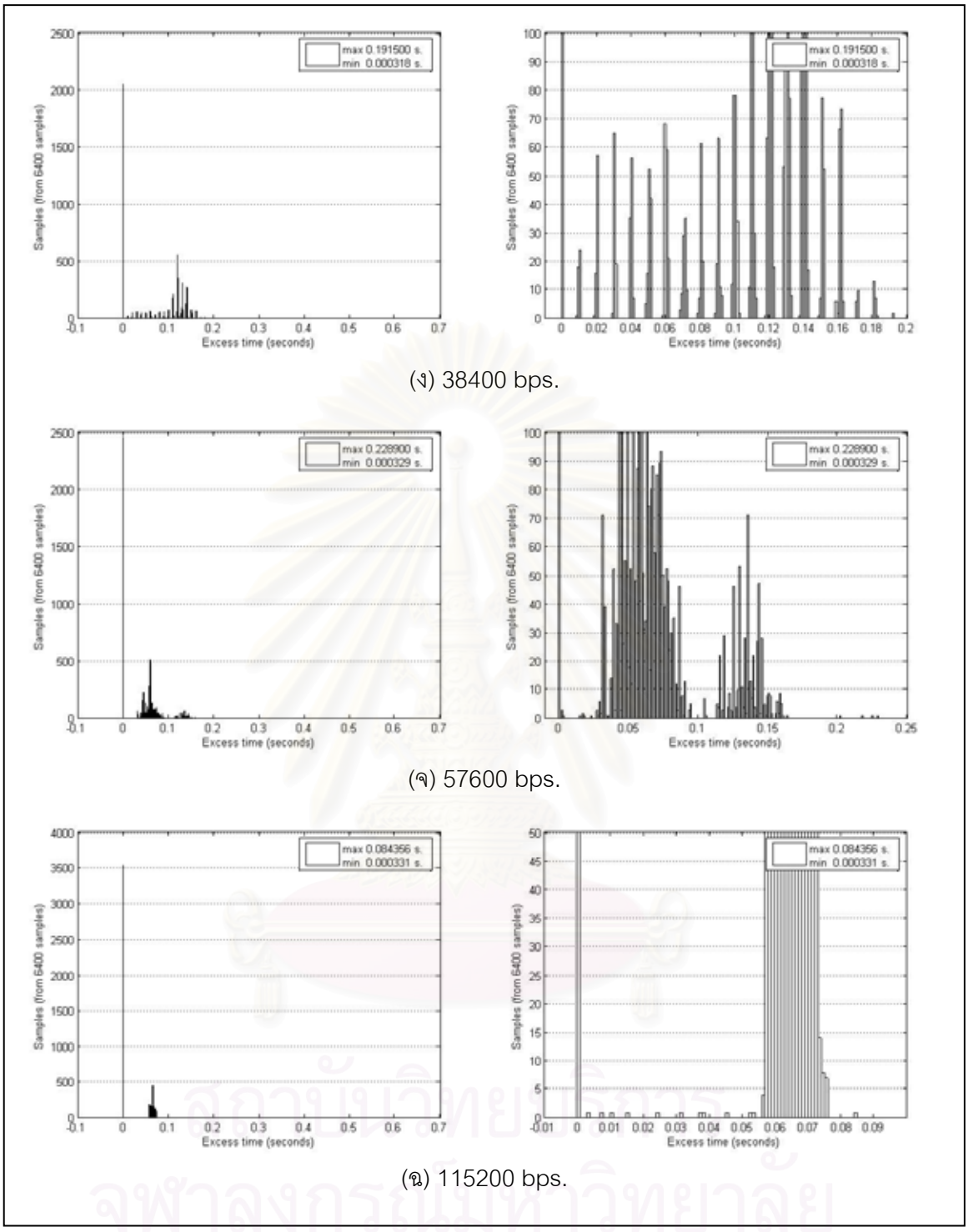


(ก) 4800 bps.

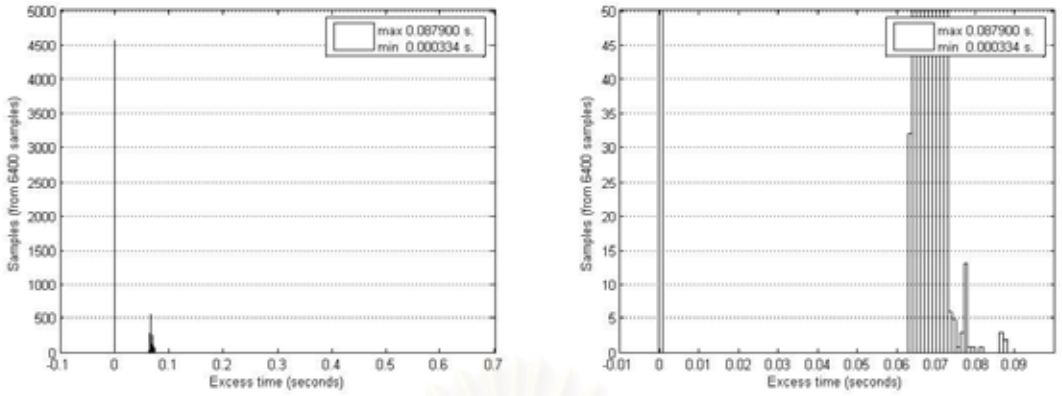
(ข) 9600 bps.

(ค) 19200 bps.

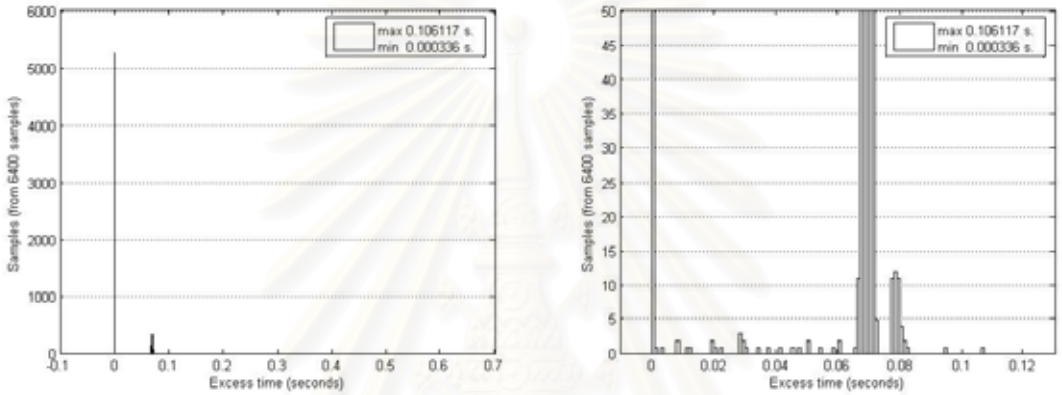
รูป 3.7 Excess time ของ delay3 ของการทดสอบแบบที่ 2 ด้วยวิธีการหยั่งสัญญาณ



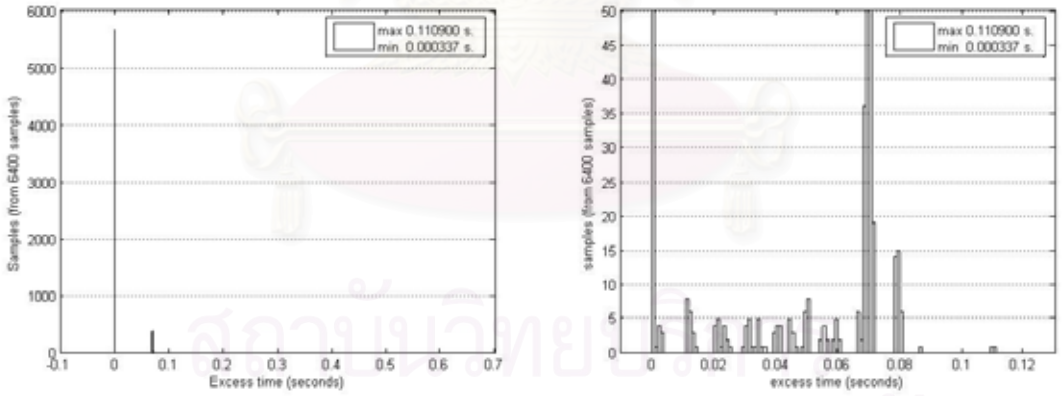
รูป 3.7 (ต่อ) Excess time ของ delay3 ของการทดสอบแบบที่ 2 ด้วยวิธีการหยั่งสัญญาณ



(ช) 230400 bps.

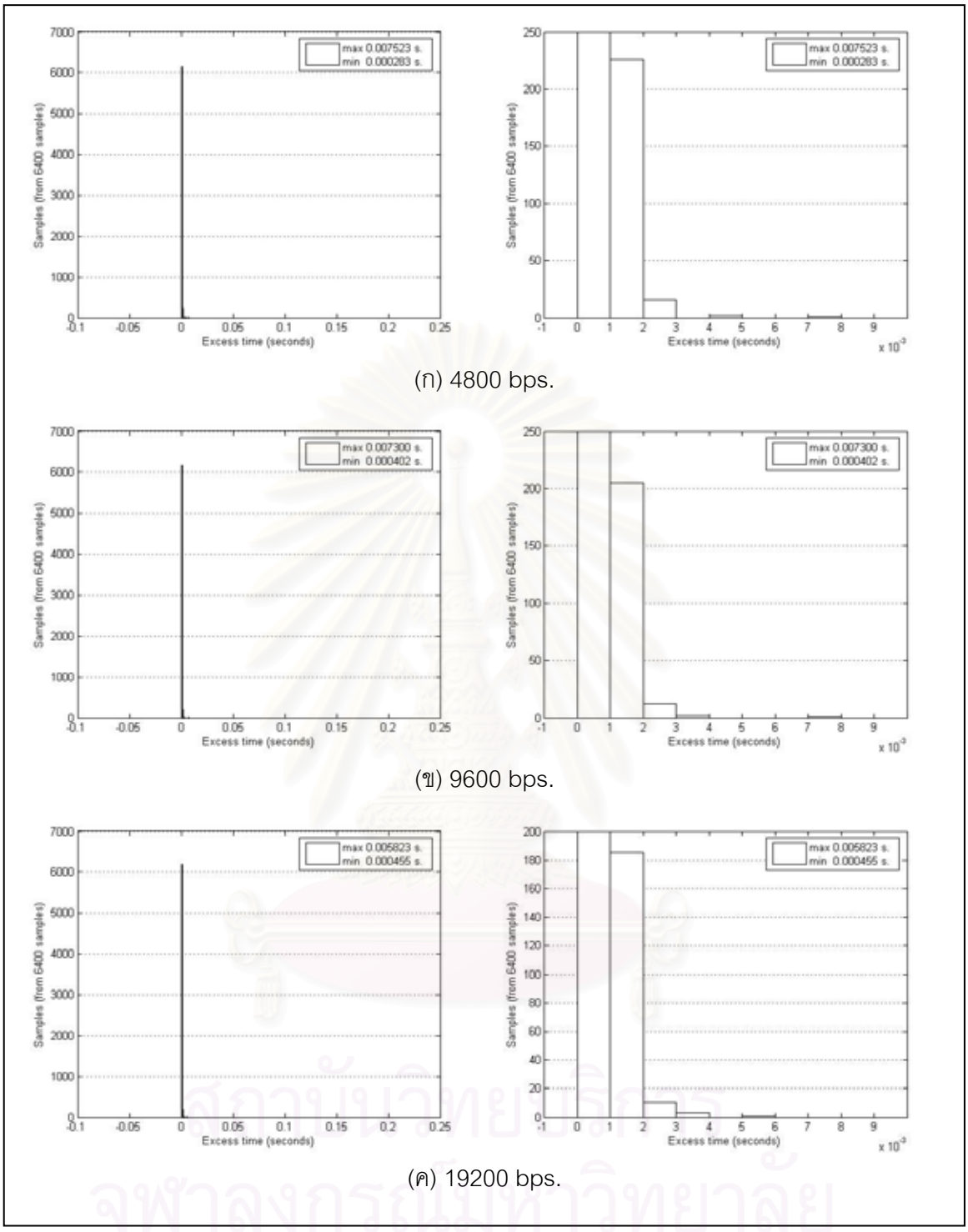


(ซ) 460800 bps.



(ฌ) 921600 bps.

รูป 3.7 (ต่อ) Excess time ของ delay3 ของการทดสอบแบบที่ 2 ด้วยวิธีการหยั่งสัญญาณ

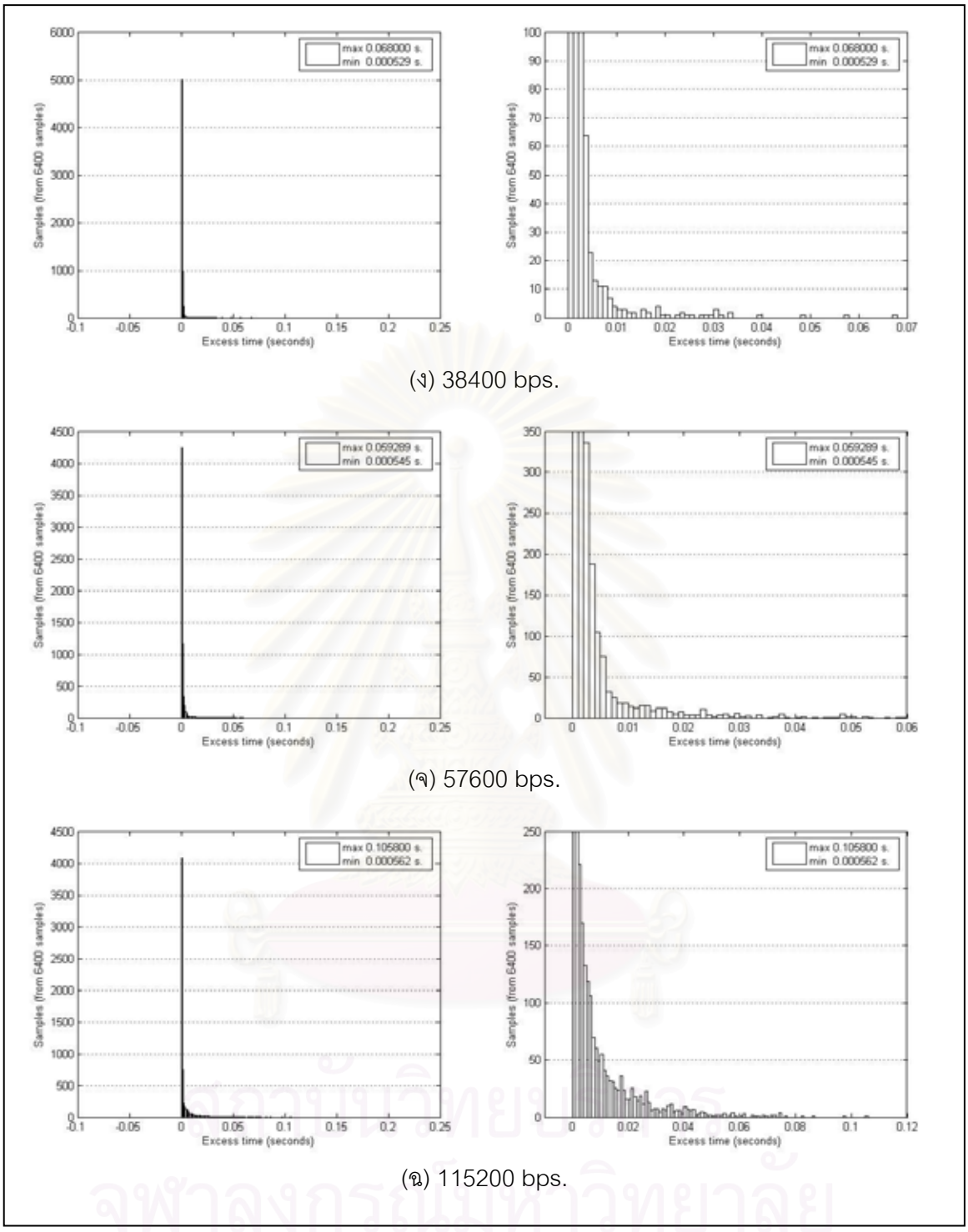


(ก) 4800 bps.

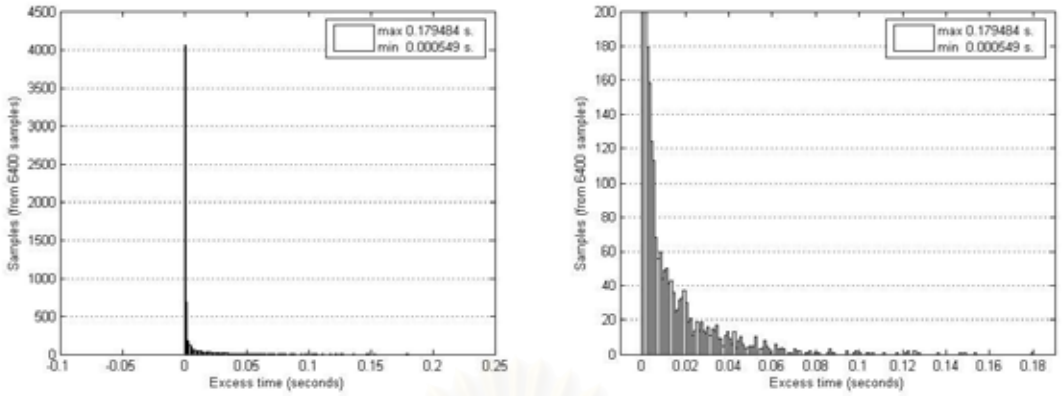
(ข) 9600 bps.

(ค) 19200 bps.

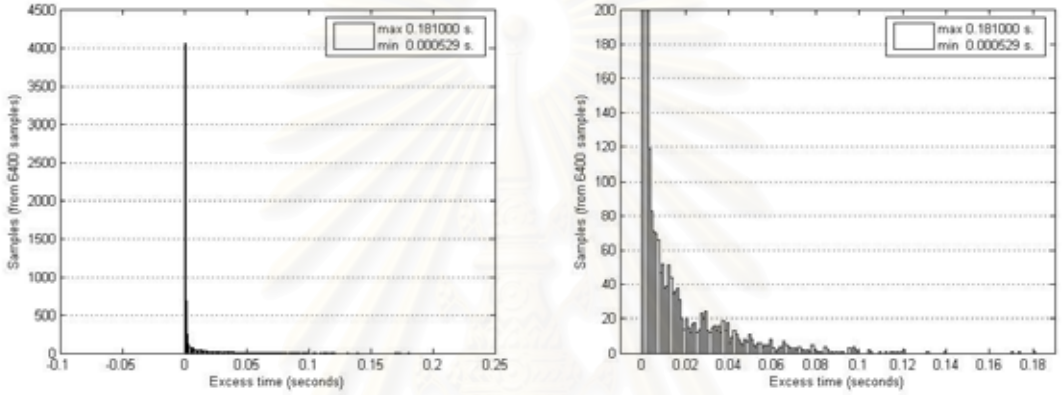
รูป 3.8 Excess time ของ delay3 ของการทดสอบแบบที่ 2 ด้วยวิธีรับส่งสัญญาณ



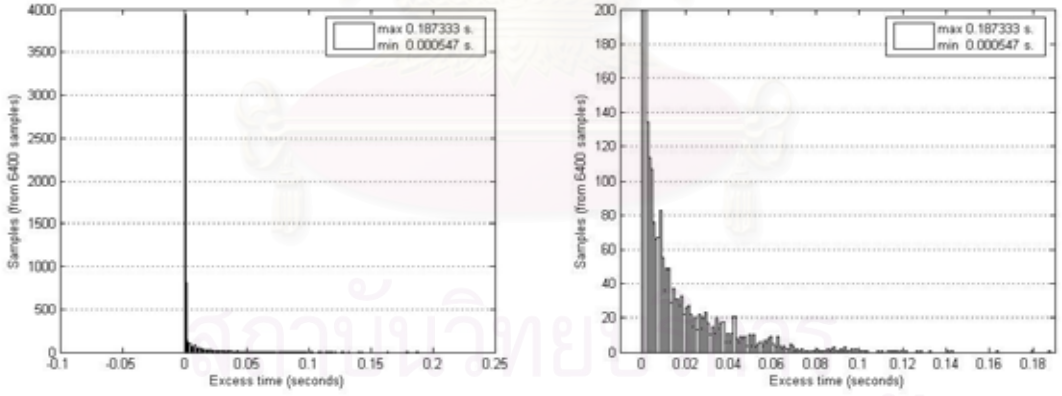
รูป 3.8 (ต่อ) Excess time ของ delay3 ของการทดสอบแบบที่ 2 ด้วยวิธีรับส่งสัญญาณ



(ช) 230400 bps.

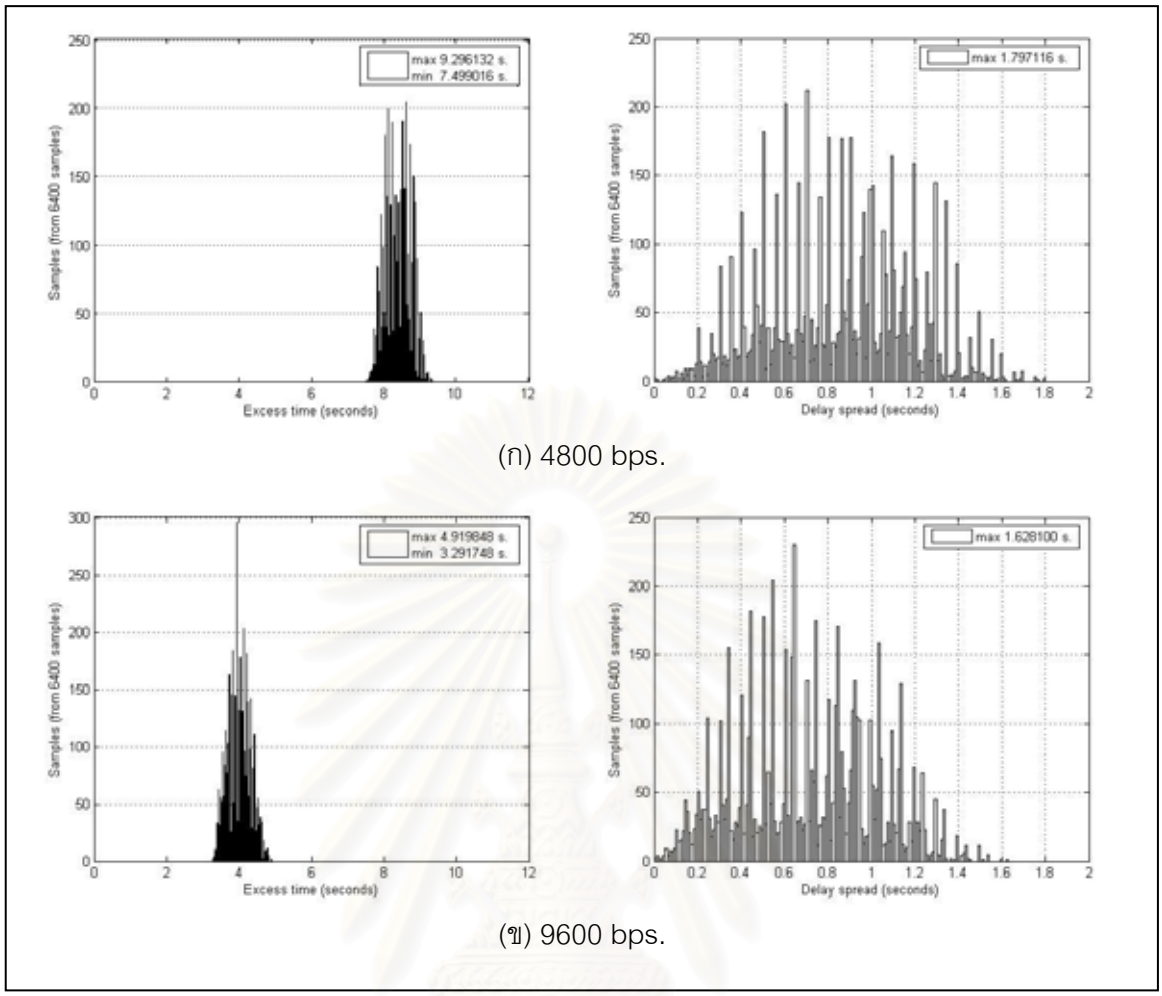


(ซ) 460800 bps.

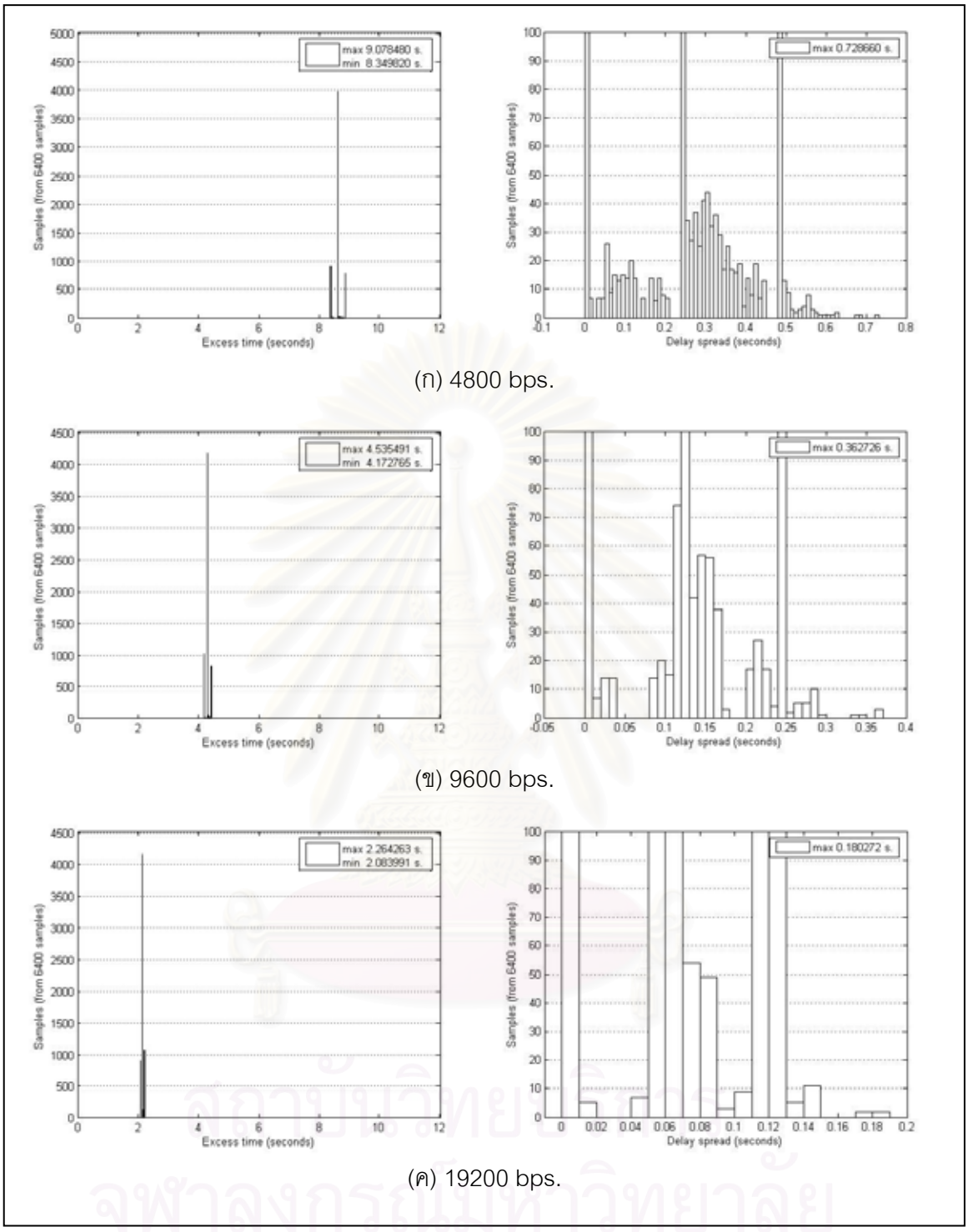


(ฌ) 921600 bps.

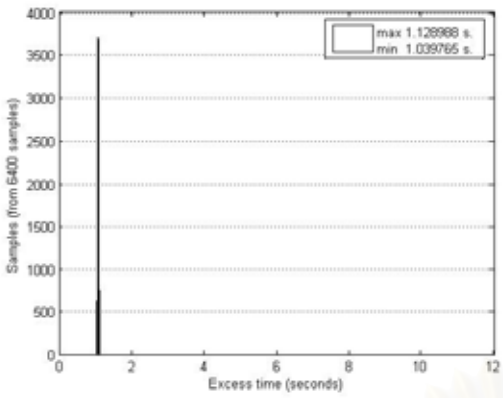
รูป 3.8 (ต่อ) Excess time ของ delay3 ของการทดสอบแบบที่ 2 ด้วยวิธีรับส่งสัญญาณ

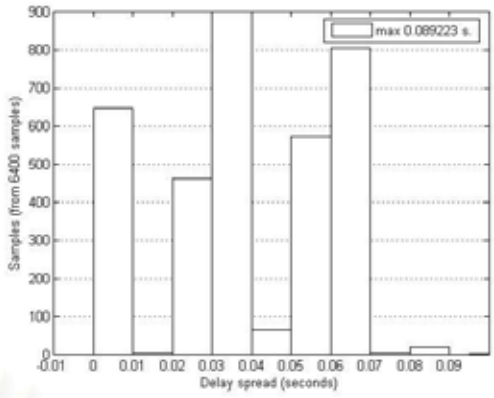


รูป 3.9 Excess time ของ delay3 ของการทดสอบแบบที่ 3 ด้วยวิธีการหยั่งสัญญาณ

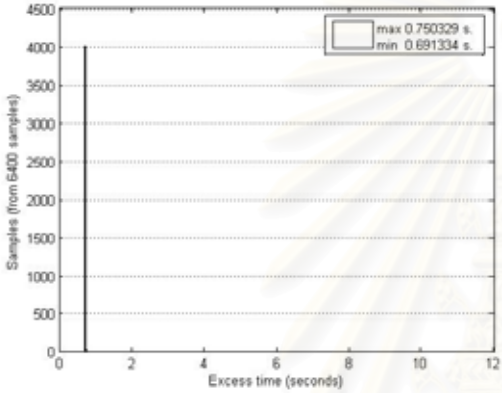


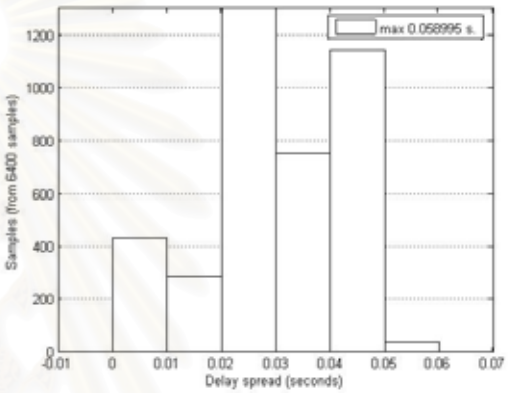
รูป 3.10 Excess time ของ delay3 ของการทดสอบแบบที่ 3 ด้วยวิธีรับส่งสัญญาณ



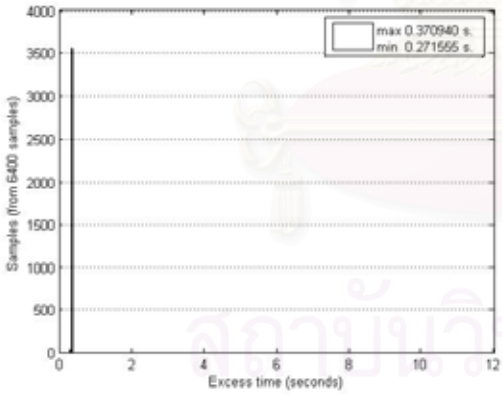


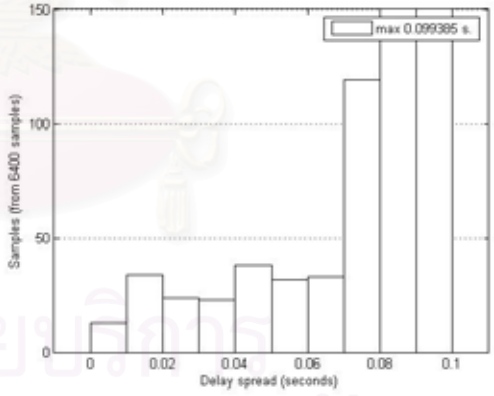
(ง) 38400 bps.





(จ) 57600 bps.





(ฉ) 115200 bps.

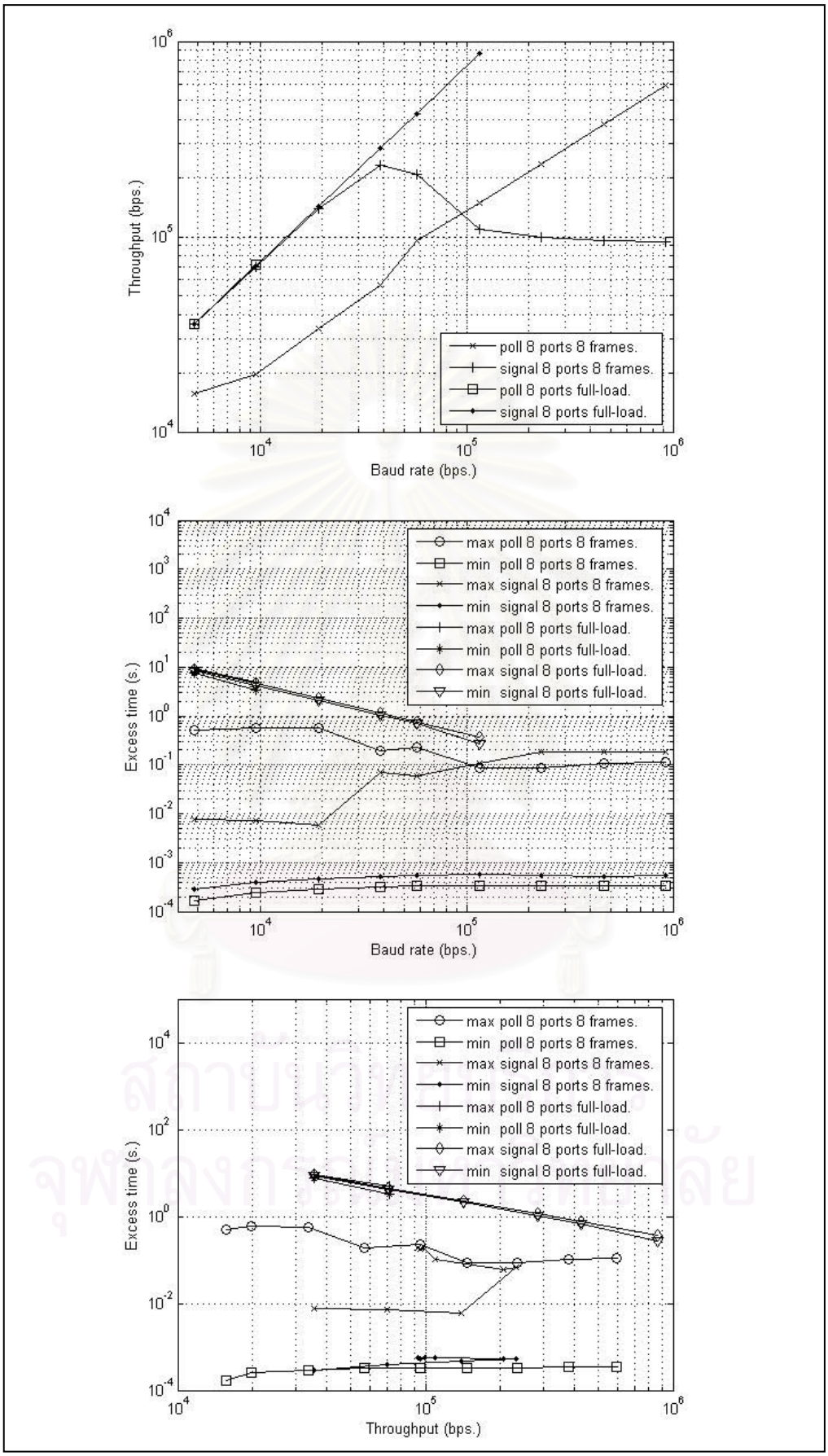
รูป 3.10 (ต่อ) Excess time ของ delay3 ของการทดสอบแบบที่ 3 ด้วยวิธีรับส่งสัญญาณ

ตาราง 3.10 ปริมาณงานของการทดสอบแบบที่ 2

BAUD rate (bps.)	POLL		SIGNAL	
	Throughput / port (bps.)	%	Throughput / port (bps.)	%
4800	1969	41.02	4436	92.42
9600	2467	25.70	8805	91.72
19200	4212	21.94	17423	90.74
38400	7048	18.35	29080	75.73
57600	11892	20.65	25947	45.05
115200	18486	16.05	13686	11.88
230400	29515	12.81	12454	5.41
460800	47289	10.26	11938	2.59
921600	74046	8.03	11691	1.27

ตาราง 3.11 ปริมาณงานของการทดสอบแบบที่ 3

BAUD rate (bps.)	POLL		SIGNAL	
	Throughput / port (bps.)	%	Throughput / port (bps.)	%
4800	4458	92.87	4437	92.43
9600	8954	93.27	8874	92.43
19200	Segmentation fault		17749	92.44
38400			35500	92.44
57600			53251	92.44
115200			108131	93.86
230400			Segmentation fault	
460800				
921600				



รูป 3.11 ปริมาณงาน

บทที่ 4

ผลการวิเคราะห์ข้อมูล

4.1 ผลการเปรียบเทียบ

4.1.1 PC/104 รุ่น PCM-3341 + PCM-3643

1. จากรูป 3.4 ค่าสูงสุดและค่าเฉลี่ย excess time ของ delay 1 โดยวิธีรับส่งสัญญาณมักจะมีค่าต่ำกว่าวิธีการหยั่งสัญญาณ แต่ในทางตรงข้ามค่าต่ำสุดของวิธีรับส่งสัญญาณมักจะมีค่าสูงกว่าวิธีการหยั่งสัญญาณ

2. จากรูป 3.5 ค่าเฉลี่ย excess time ของ delay 1 ของการทดสอบกับช่องทาง 1-4 มักจะต่ำกว่าการทดสอบกับช่องทาง 5-8

3. จากรูป 3.6 ค่าเฉลี่ย excess time ของ delay 1 ที่ใช้วิธีการหยั่งสัญญาณที่การแบ่งเวลา 10 มิลลิวินาทีจะมีค่าต่ำกว่าที่การแบ่งเวลา 100 มิลลิวินาที ขณะที่การแบ่งเวลา 10 หรือ 100 มิลลิวินาทีดูเหมือนจะไม่มีผลต่อวิธีรับส่งสัญญาณมากนัก

4. จากตาราง 3.3 3.5 3.7 3.9 การทดสอบทำได้ถึงอัตราบอด 300 และ 600 ก่อนจะล้มเหลวด้วย Unknown interrupt or fault at EIP ...

4.1.2 RISC base embedded computer รุ่น UC-7408

1. จากรูป 3.7 excess time ของ delay 3 ของการทดสอบแบบที่ 2 ด้วยวิธีการหยั่งสัญญาณสามารถแบ่ง excess time ได้เป็น 2 กลุ่มคือกลุ่มที่ 1 มีค่าเข้าใกล้ 0 ส่วนที่เหลือจัดเป็นกลุ่มที่ 2 โดยที่อัตราบอดต่ำกลุ่ม 1 มีค่าต่ำและกลุ่ม 2 มีการกระจายตัวมาก ส่วนที่อัตราบอดสูงกลุ่ม 1 จะมีค่าสูงขึ้นและกลุ่ม 2 มีค่าต่ำลงและค่อย ๆ ลู่เข้าหาจุดเวลา 70 มิลลิวินาที

2. จากรูป 3.8 excess time ของ delay 3 ของการทดสอบแบบที่ 2 ด้วยวิธีรับส่งสัญญาณได้ผลว่า excess time เกือบทั้งหมดมีค่าเข้าใกล้ 0 แต่ที่อัตราบอดสูงขึ้นจะมีบางส่วนที่ค่อย ๆ ออกห่างจาก 0 ไปทางด้านเวลาบวกเพิ่มขึ้น

3. จากรูป 3.9 excess time ของ delay 3 ของการทดสอบแบบที่ 3 ด้วยวิธีการหยั่งสัญญาณปรากฏว่า excess time มีการกระจายตัวคล้ายการกระจายตัวแบบปกติรอบจุดเวลา 8.5 และ 4.0 วินาทีตามลำดับ

4. จากรูป 3.10 excess time ของ delay 3 ของการทดสอบแบบที่ 3 ด้วยวิธีรับส่งสัญญาณปรากฏว่า excess time จับกลุ่มกันที่จุดเวลา 40960/อัตราบอด วินาที

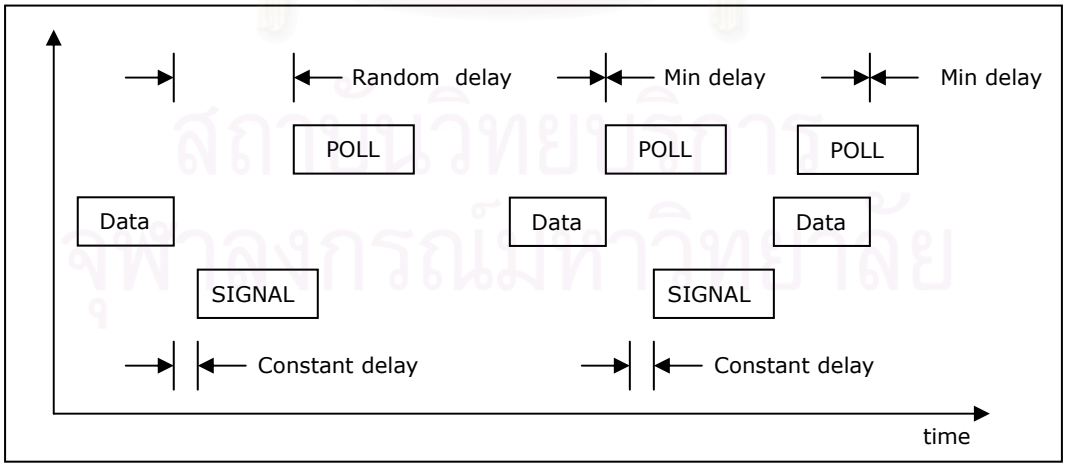
5. จากรูป 3.11 ปริมาณงานในสภาวะภาระงานปกติของวิธีรับส่งสัญญาณมีค่าเพิ่มขึ้นแล้วลดลง ส่วนในสภาวะภาระงานสูงสุดปริมาณงานโดยวิธีการหยั่งสัญญาณทำงานได้ถึงอัตราบอด 9600 บิตต่อวินาทีและวิธีรับส่งสัญญาณทำได้ถึงอัตราบอด 115200 บิตต่อวินาทีก่อนระบบจะล้มเหลวด้วย Segmentation fault

4.2 ผลการวิเคราะห์

ในหัวข้อนี้จะวิเคราะห์เป็นรายข้อให้สอดคล้องกับผลการเปรียบเทียบในหัวข้อที่ผ่านมา

4.2.1 PC/104 รุ่น PCM-3341 + PCM-3643

1. สาเหตุที่ค่าสูงสุดและค่าเฉลี่ย excess time ของ delay 1 โดยวิธีรับส่งสัญญาณมักจะมีค่าต่ำกว่าวิธีการหยั่งสัญญาณก็เพราะวิธีรับส่งสัญญาณจะไปเรียกกระบวนการที่เกี่ยวข้องกับไบต์ข้อมูลนั้น ๆ ให้มาดำเนินการกับไบต์ได้ทันทีจากการรับส่งสัญญาณ ซึ่งตรงข้ามกับวิธีการหยั่งสัญญาณที่กระบวนการที่เกี่ยวข้องกับไบต์ข้อมูลนั้น ๆ จะเข้ามาดำเนินการกับไบต์ได้ก็ต่อเมื่อถึงเวลาดำเนินงานตามการสลับเวลาของเคอร์เนลซึ่งอยู่ในลักษณะของการเข้าคิวที่จะแซงคิวไม่ได้ ส่วนสาเหตุที่ค่าต่ำสุดของ delay 1 โดยวิธีรับส่งสัญญาณมักจะมีค่าสูงกว่าวิธีการหยั่งสัญญาณก็เพราะกระบวนการที่ทำงานด้วยวิธีรับส่งสัญญาณจะทำงานได้ก็ต่อเมื่อได้รับการกระตุ้นจากสัญญาณแล้วถึงจะทำงาน ทำให้เกิดระยะเวลาหน่วงช่วงหนึ่งที่ใช้ไปในการรับส่งสัญญาณ ในขณะที่กระบวนการที่ทำงานด้วยวิธีการหยั่งสัญญาณจะเข้ามาจัดการไบต์ข้อมูลอยู่ตลอดเวลาในช่วงเวลาที่ได้รับอนุญาตให้ดำเนินงานได้จากการจัดเวลาของเคอร์เนลจึงเป็นไปได้ที่จะมีโอกาสเข้าไปจัดการไบต์ข้อมูลในขณะที่ไบต์ข้อมูลเข้ามาครบพอดีโดยไม่มีระยะเวลาหน่วงเหมือนกับวิธีรับส่งสัญญาณดังแสดงในรูป 4.1



รูป 4.1 เวลาหน่วงในการดำเนินการกับไบต์ข้อมูลด้วยวิธีรับส่งสัญญาณกับวิธีการหยั่งสัญญาณ

2. สาเหตุที่ excess time ของ delay 1 จากการทดสอบด้วยช่องทาง 1-4 มักจะมีค่าต่ำกว่าการทดสอบด้วยช่องทาง 5-8 ก็เพราะช่องทาง 1-4 อยู่บนแผงหลักและใช้ไอออาร์คิวแยกกันทุกช่องทาง

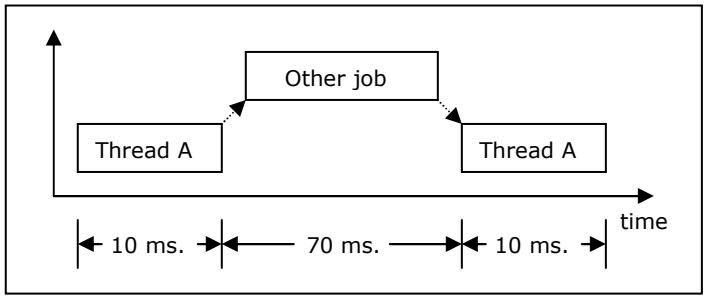
ในขณะที่ช่องทาง 5-8 เชื่อมต่อกับแผงหลักผ่านบัสไอเอสเอและทั้ง 4 ช่องทางยังใช้ไออาร์คิวร่วมกันทำให้เวลาดำเนินการของช่องทาง 5-8 มักจะมีค่าสูงกว่าช่องทาง 1-4

3. สาเหตุที่การแบ่งเวลาสั้นหรือยาวมีผลต่อวิธีการหยั่งสัญญาณก็เพราะค่าการแบ่งเวลาเหล่านั้นจะแสดงถึงความถี่และระยะเวลาที่กระบวนการจะเข้าไปจัดการกับไบต์ข้อมูลว่ามีบ่อยหรือมากน้อยเพียงใดเพราะในการหยั่งสัญญาณจะต้องรอเวลาดำเนินการให้เป็นไปตามคิวที่เคอร์เนลจัดให้ โดยการแบ่งเวลา 10 มิลลิวินาทีย่อมทำให้กระบวนการที่เกี่ยวข้องกับไบต์ข้อมูลได้มีโอกาสเข้ามาดำเนินการกับไบต์ข้อมูลถี่กว่าการแบ่งเวลา 100 มิลลิวินาทีแม้ว่าในแต่ละครั้งจะมีเวลาทำงานที่สั้นกว่าแต่ก็ถือว่าเพียงพอแล้วเมื่อดูจาก excess time ที่ส่วนใหญ่มีค่าต่ำกว่า 10 มิลลิวินาทีกล่าวคือระบบสามารถจัดการกับ 1 กรอบความให้เสร็จได้ภายใน 10 มิลลิวินาทีหรือ 1 ช่วงการแบ่งเวลา ในทางตรงข้ามการแบ่งเวลาแทบจะไม่มีผลต่อกระบวนการที่ใช้วิธีรับส่งสัญญาณเพราะกระบวนการดังกล่าวจะทำงานก็ต่อเมื่อได้รับการกระตุ้นจากสัญญาณแล้วเท่านั้น

4. สาเหตุของความล้มเหลวเมื่อทดสอบกับช่องทาง 4-8 เป็นเพราะช่องทางทั้ง 4 ใช้ไออาร์คิวร่วมกันบนบัสแบบไอเอสเอที่ใช้การขัดจังหวะแบบขอบทำให้ตัวควบคุมการขัดจังหวะเกิดความสับสนหรืออาจเป็นเพราะมีสัญญาณขอขัดจังหวะจากช่องบางสัญญาณไม่ถูกตรวจจับหรือสูญหายไปนั้นเอง

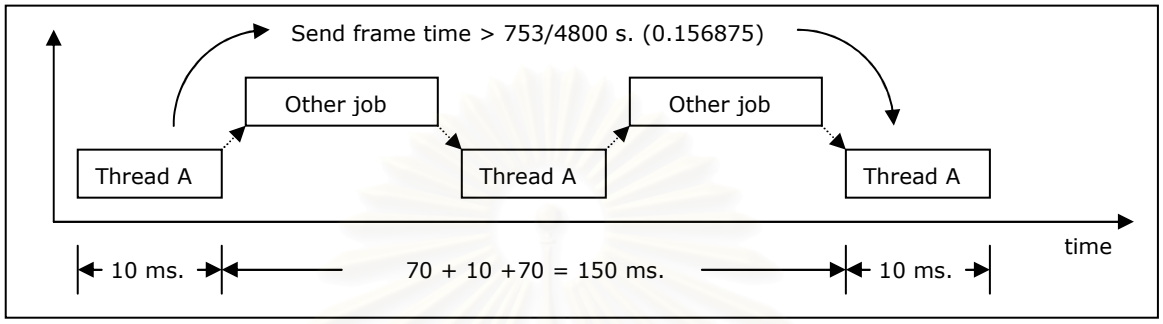
4.2.2 RISC base embedded computer รุ่น UC-7408

1. จากการสังเกตุข้อมูลดิบที่อัตราบอด 921600 บิตต่อวินาทีทำให้ประมาณการได้ว่าการแบ่งเวลาของเคอร์เนลใน UC-7408 ให้แต่ละสายโยงใยอยู่ที่ประมาณ 10 มิลลิวินาทีโดยมีระยะห่างก่อนที่จะได้รับอนุญาตให้ประมวลผลได้อีกครั้งอยู่ที่ประมาณ 70 มิลลิวินาทีดังรูป 4.2 ซึ่งข้อมูลในกลุ่ม 1 คือกรอบความที่ส่งออกไปแล้วรับเข้ามาประมวลผลภายในช่วงการแบ่งเวลา 10 มิลลิวินาทีและจากการสังเกตุผลปรากฏว่าช่วงการแบ่งเวลา 10 มิลลิวินาทีสายโยงใยสามารถส่งและรับกรอบความได้ 8 กรอบความ ส่วนข้อมูลกลุ่ม 2 คือกรอบความที่ส่งออกไปช่วงปลายของการแบ่งเวลาและกรอบความนั้นต้องรอเวลาอีกประมาณ 70 มิลลิวินาทีจึงจะได้ประมวลผลทำให้ที่อัตราบอด 921600 บิตต่อวินาทีอัตราส่วนข้อมูลกลุ่ม 1 ต่อกลุ่ม 2 อยู่ที่ประมาณ 1 ต่อ 8 และข้อมูลกลุ่ม 2 ก็อยู่ที่บริเวณ 70 มิลลิวินาที



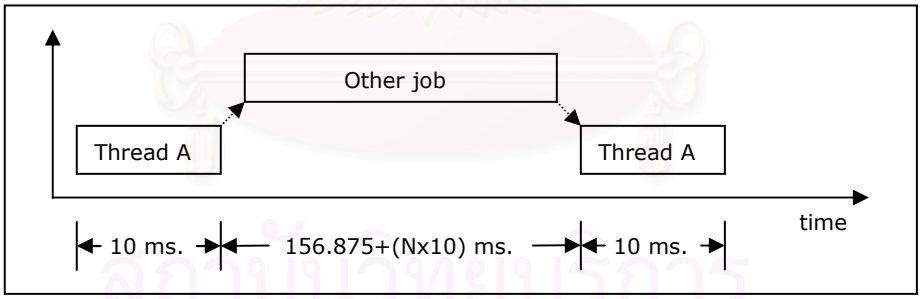
รูป 4.2 การทำงานของสายโยงใยที่อัตราบอด 921600 บิตต่อวินาทีด้วยวิธีการหยั่งสัญญาณ

ส่วนที่อัตราบอดต่ำนั้นจังหวะการส่งและรับจะต้องทำต่างช่วงของการแบ่งเวลาเช่นที่อัตราบอด 4800 บิตต่อวินาทีต้องใช้เวลาประมาณ 753/4800=0.156875 วินาทีต่อการส่ง 1 กรอบความซึ่งมากกว่าช่วงของการแบ่งเวลา 10 มิลลิวินาทีและหากใช้จังหวะการทำงานเหมือนกับที่อัตราบอด 921600 บิตต่อวินาทีมีก็จะได้ผลดังรูป 4.3 ซึ่งเห็นได้ว่าระหว่างการส่งจนถึงการรับจะต้องมี 1 ช่วงเวลาที่ต้องทำงานเปล่าโดยที่ไม่มีกรอบความในการทำงานเลย



รูป 4.3 ระยะเวลาการทำงานของสายโยงใยที่อัตราบอด 4800 บิตต่อวินาทีด้วยวิธีการหยั่งสัญญาณ

ถ้าการส่งและรับกรอบความที่อัตราบอด 4800 บิตต่อวินาทีมีลักษณะเหมือนในรูป 4.3 จะได้ว่า excess time มีค่าเข้าใกล้ 0 แต่อย่างไรก็ตามการสลับเวลาทำงานของเคอร์เนลให้แต่ละสายโยงใยนั้นไม่แน่นอนทำให้ excess time ที่ได้มีตั้งแต่ค่าเข้าใกล้ 0 ไปจนเกือบถึง 600 มิลลิวินาทีและเมื่อสังเกตุผลการทดสอบที่อัตราบอดต่ำด้วยวิธีการหยั่งสัญญาณจะเห็นว่ายอดของ excess time มีช่วงห่างกันประมาณ 10 มิลลิวินาทีซึ่งช่วงห่าง 10 มิลลิวินาทีนี้มาจากการแบ่งเวลาของเคอร์เนลดังรูป 4.4

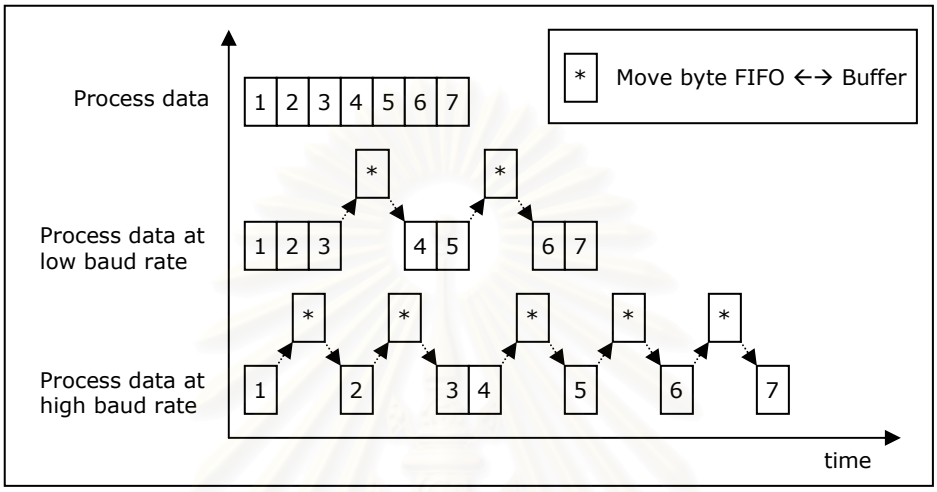


รูป 4.4 ระยะเวลาการทำงานของสายโยงใยด้วยวิธีการหยั่งสัญญาณ

กรณี N=0 จะได้ว่า excess time เข้าใกล้ 0

กรณี N≠0 จะได้ว่า excess time เพิ่มขึ้นทีละประมาณ 10 มิลลินาที ซึ่งในกรณีอัตราบอดต่ำจะเห็นได้ว่า excess time มีค่าได้มากถึง 581.300 มิลลิวินาทีนั้นแปลว่าหลังจากสายโยงใยส่งกรอบความออกไปแล้วเคอร์เนลแบ่งเวลาให้สายโยงใยอื่นสลับกันทำงานยาวนานมากสุดถึง 581.300-156.875=424.425 มิลลิวินาทีหรือคิดเป็นค่า N ได้ประมาณ 42 ถึง 43 ก่อนจะให้สายโยงใยดังกล่าวกลับมาทำงานอีกครั้งหนึ่ง

2. ที่อัตราบอดสูงไบต์ข้อมูลที่เข้าออกตามช่องทางต่าง ๆ ย่อมเข้ามาหรือออกจากไฟโฟได้เร็วกว่าที่อัตราบอดต่ำทำให้ที่อัตราบอดสูงซีพียูจะถูกเรียกให้ไปย้ายไบต์ข้อมูลจากไฟโฟมาเก็บที่บัฟเฟอร์หรือจากบัฟเฟอร์ไปที่ไฟโฟบ่อยกว่าที่อัตราบอดต่ำ ทำให้เวลาประมวลผลต่อ 1 กรอบความมีค่าสูงขึ้นเพราะซีพียูต้องสลับไปย้ายไบต์ข้อมูลบ่อย ๆ ดังรูป 4.5 ส่งผลให้ที่อัตราบอดสูงจึงมี excess time บางส่วนมีค่าเพิ่มขึ้น



รูป 4.5 การสลับการทำงานของซีพียูระหว่างการประมวลผลกับการย้ายไบต์ระหว่างบัฟเฟอร์กับไฟโฟ

3. สาเหตุที่ excess time ของ delay 3 ในการทดสอบแบบที่ 3 ด้วยวิธีการหยั่งสัญญาณมีการกระจายตัวคล้ายการกระจายแบบปกติเพราะการทำงานด้วยการหยั่งสัญญาณจะทำงานได้ก็ต่อเมื่อถึงจังหวะเวลาที่เคอร์เนลจัดให้ดังรูป 4.1 ซึ่งมีระยะเวลาไม่แน่นอนเช่นหากกรอบความเข้ามาในจังหวะที่ผู้รับอยู่ในจังหวะทำงานอยู่กรอบความนั้นก็จะได้รับการประมวลผลเร็ว แต่หากเข้ามาในช่วงที่ผู้รับไม่ได้อยู่ในจังหวะทำงานกรอบความนั้นก็จะได้รับการประมวลผลช้า ส่วนสาเหตุที่การกระจายมีจุดศูนย์กลางอยู่ที่เวลาประมาณ 8.5 และ 4.0 วินาทีก็เพราะว่าการจับเวลาผู้ส่งจะเริ่มจับตั้งแต่ก่อนส่งกรอบความเข้าไปในบัฟเฟอร์ขาส่งซึ่งในการทดสอบแบบที่ 3 นี้มีการส่งกรอบเข้าไปในบัฟเฟอร์ขาส่งตลอดทำให้ข้อมูลต้องเข้าไปต่อคิวในบัฟเฟอร์ก่อนที่จะถูกส่งออกไปตามช่องทางข้อมูลอนุกรมซึ่งเวลาที่ต้องรอนี้มีค่าประมาณ จำนวนบิตในบัฟเฟอร์/อัตราบอด ซึ่งหากคิดว่าบัฟเฟอร์มีความจุเท่ากับ 4096 ไบต์หรือคิดเป็น 40960 บิต (ในการทดสอบ 1 ไบต์มี 10 บิต) ก็จะได้เวลาที่ต้องรอเท่ากับ 40960/4800 = 8.53 วินาทีหรือ 40960/9600 = 4.27 วินาที ซึ่งเห็นได้ว่ามีค่าใกล้เคียงกับจุดศูนย์กลางของการกระจายแต่ในการทดสอบนี้ผู้ส่งทำงานอยู่บนเครื่องเดียวกับผู้รับและกระบวนการรับก็ใช้วิธีการหยั่งสัญญาณซึ่งทุก ๆ กระบวนการรับจะทำงานเต็มช่วงการแบ่งเวลาที่ได้รับการจัดสรรจากเคอร์เนลทำให้ผู้ส่งมีสัดส่วนของเวลาการทำงานประมาณ 1 ใน 9 ส่วน (ส่ง 1 ส่วน รับ 8 ส่วน) และในช่วงที่ผู้ส่งไม่ได้ส่งไบต์ข้อมูลเข้าบัฟเฟอร์ ไบต์ที่ค้างในบัฟเฟอร์ก็ทยอยออกไปตามช่องทางได้นาน 8 ใน 9 ส่วนเมื่อถึงเวลาที่ผู้ส่งเข้ามาส่งกรอบความจำนวนไบต์ในบัฟเฟอร์จึงเหลือน้อยเมื่อเทียบกับจำนวนไบต์ที่เหลือในบัฟเฟอร์ในวิธีรับส่งสัญญาณทำให้เวลาต่ำสุดของวิธีการหยั่งสัญญาณมีค่าน้อยกว่า 40960/อัตรา

บอด เช่นที่อัตราบอด 4800 บิตต่อวินาทีค่าต่ำสุดอยู่ที่ 7.499016 วินาทีหรือคิดคร่าว ๆ ได้ว่ากรอบความดังกล่าวเข้ามาต่อคิวในบัฟเฟอร์โดยมีบิตก่อนหน้าไม่เกิน 7.499016x4800 = 35995 บิตหรือประมาณ 3600 ไบต์ ส่วนที่อัตราบอด 9600 บิตต่อวินาทีมีบิตอยู่ไม่เกิน 3.291748x9600 = 31600 บิตหรือประมาณ 3160 ไบต์จากความจุบัฟเฟอร์ 4096 ไบต์ ส่วนค่าสูงสุดคงมาจากกรอบความนั้นเข้ามาต่อคิวในจังหวะที่มีไบต์ข้อมูลในบัฟเฟอร์ขาส่งเต็มหรือเกือบเต็มเช่นที่อัตราบอด 4800 บิตต่อวินาทีก็ต้องรอประมาณ 8.53 ส่วนที่อัตราบอด 9600 บิตต่อวินาทีก็ต้องรอประมาณ 4.27 วินาทีรวมกับการรอเวลาที่จะประมวลผลจากการจัดสรรของเคอร์เนลที่นานที่สุดเช่นประมวลผลกรอบความไม่เสร็จจนต้องนำไปประมวลต่อในช่วงการทำงานถัดไป

4. สาเหตุที่ excess time ของ delay 3 ในการทดสอบแบบที่ 3 ด้วยวิธีรับส่งสัญญาณจับกลุ่มกันบริเวณ 40960/อัตราบอด เป็นเพราะผู้รับในวิธีการรับส่งสัญญาณจะทำงานก็ต่อเมื่อได้รับการกระตุ้นจากสัญญาณที่เหมาะสมเท่านั้นและในขณะที่ไม่มีสัญญาณมากระตุ้นผู้รับจะไม่ทำงาน ทำให้ทางผู้ส่งกรอบความสามารถส่งกรอบความเข้าสู่บัฟเฟอร์ขาส่งได้เกือบตลอดเวลาที่สามารถส่งได้เป็นผลให้จำนวนไบต์ก่อนหน้าในบัฟเฟอร์มีจำนวนมากตัวอย่างเช่นที่อัตราบอด 4800 บิตต่อวินาทีค่าต่ำสุดอยู่ที่ 8.349820 วินาทีหรือคิดคร่าว ๆ ได้ว่ากรอบความดังกล่าวเข้ามาต่อคิวในบัฟเฟอร์โดยมีบิตอยู่ก่อนหน้าไม่เกิน 8.349820x4800 = 40079 บิตหรือประมาณ 4008 ไบต์ ส่วนที่อัตราบอด 9600 บิตต่อวินาทีมีบิตอยู่ก่อนหน้าไม่เกิน 4.172765x9600 = 40059 บิตหรือประมาณ 4006 ไบต์จากความจุบัฟเฟอร์ 4096 ไบต์ รวมกับการทำงานด้วยวิธีรับส่งสัญญาณจะมีเวลาประมวลผลที่ค่อนข้างคงที่ตามรูป 4.1 ทำให้ excess time มีค่าประมาณ (40000/อัตราบอด)+เวลาประมวลผลค่าหนึ่งส่งผลให้ excess time ที่ได้จับกลุ่มกันมากกว่าวิธีการหยั่งสัญญาณ

5. สาเหตุที่ปริมาณงานในสภาวะภาระงานปกติของวิธีรับส่งสัญญาณมีค่าเพิ่มขึ้นแล้วลดลงอาจเป็นเพราะในช่วงบอดต่ำปริมาณงานจะขึ้นกับอัตราบอด โดยที่อัตราบอดต่ำปริมาณงานก็จะน้อยเพราะเวลาส่วนใหญ่ที่ใช้ไปจะอยู่ที่การรับส่งและปริมาณงานนี้ก็จะสูงขึ้นเรื่อย ๆ ตามอัตราบอดที่เพิ่มขึ้นจนถึงจุดหนึ่งก็จะค่อย ๆ ลดลงนั้นอาจเป็นเพราะเมื่อข้อมูลถูกรับส่งมากขึ้นการขอขัดจังหวะก็มากขึ้นทำให้เวลาที่ใช้ในการประมวลผลมากตามไปด้วยและเมื่อเวลาที่ใช้ประมวลผลเพิ่มตามจำนวนของการขอขัดจังหวะก็ทำให้ปริมาณงานในช่วงอัตราบอดสูง ๆ จะขึ้นอยู่กับเวลาที่ใช้ประมวลผลและเมื่อประมวลผลได้ช้าลงปริมาณงานก็ลดลงและในที่สุดปริมาณงานก็จะลู่เข้าหาจุดสมดุลซึ่งจากรูป 3.11 คือลู่เข้าสู่ปริมาณงาน 90000 บิตต่อวินาที ในขณะที่ปริมาณงานของวิธีการหยั่งสัญญาณไม่ลดลงเป็นเพราะการหยั่งสัญญาณทำงานตามการแบ่งเวลาไม่ได้ทำงานตามสัญญาณขอขัดจังหวะส่วนปริมาณงานของการรับส่งสัญญาณที่สภาวะภาระงานสูงสุดไม่ลดลงเป็นเพราะมีการรับส่งข้อมูลอยู่ตลอดเวลาโดยข้อมูลเหล่านั้นคือข้อมูลที่ค้างอยู่ในบัฟเฟอร์ 4096 ไบต์นั้นเอง ส่วนสาเหตุของ Segmentation fault นี้ทางผู้ทดสอบยังหาสาเหตุไม่พบ

บทที่ 5

สรุปผลการทดสอบ และข้อเสนอแนะ

วิทยานิพนธ์ฉบับนี้เป็นการศึกษาสมรรถนะของระบบปฏิบัติการลินุกซ์บนระบบฝังตัวเพื่อนำมาทำเป็นตัวควบคุมการสื่อสารของเครื่องบินแบบไม่มีคนบังคับอยู่บนเครื่อง (ยูเอวี) และโดยปกติตัวควบคุมการสื่อสารนี้จะต้องรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรม 8 ช่องพร้อมกันที่อัตราบอด 9600 บิตต่อวินาทีและมีปริมาณงาน 38400 บิตต่อวินาที ซึ่งในวิทยานิพนธ์ฉบับนี้ได้มีการทดสอบกับฮาร์ดแวร์ 2 แบบคือพีซี/104 และ RISC based embedded computer ที่มีข้อแตกต่างกันคือช่องทางข้อมูลอนุกรมของพีซี/104 จะอยู่บนบัสแบบไอเอสเอที่ใช้สัญญาณขอขัดจังหวะแบบขอบซึ่งจะใช้ไอ อาร์คิวร่วมได้ก็ต่อเมื่อมีโปรแกรมขับที่เหมาะสม ส่วนช่องทางข้อมูลอนุกรมของ RISC คอมพิวเตอร์ที่ใช้ทดสอบจะอยู่บนบัสแบบพีซีไอที่ใช้สัญญาณขอขัดจังหวะแบบระดับที่ใช้ไออาร์คิวร่วมได้ดี และเนื่องจากระบบฝังตัวนี้ต้องรับส่งข้อมูลจากหลายช่องทาง ดังนั้นงานที่ทำจึงเป็นแบบหลายงานทำคู่ขนานสลับกันไป ผู้ทดสอบจึงได้เลือกให้ฮาร์ดแวร์ทำงานแบบมีลินุกซ์เป็นระบบปฏิบัติการและได้เขียนโปรแกรมเป็นแบบสายโยงใย ซึ่งการรับส่งจะใช้กรอบความยาว 70 ไบต์ที่ใกล้เคียงกับการใช้งานจริงเพื่อทำการจับเวลาและนำไปวัดประสิทธิภาพ โดยได้แบ่งการทดสอบออกเป็น 2 ประเภทคือใช้กรรมวิธีการหยั่งสัญญาณและใช้กรรมวิธีรับส่งสัญญาณโดยได้แบ่งการทดสอบออกเป็น 3 ระดับคือ

1. ที่ภาระงานปกติคือระบบสามารถรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรมพร้อมกันทั้ง 8 ช่องที่อัตราบอด 9600 บิตต่อวินาทีและมีปริมาณงาน 38400 บิตต่อวินาที โดยระยะเวลาการประมวลผลยังคงยอมรับได้

2. ที่ภาระงานสูงสุดคือระบบสามารถรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรมพร้อมกันทั้ง 8 ช่องอยู่ตลอดเวลาที่อัตราบอด 9600 บิตต่อวินาที

3. หาขีดจำกัดของระบบเมื่อรับส่งข้อมูลผ่านช่องทางข้อมูลอนุกรมพร้อมกันทั้ง 8 ช่อง

5.1 สรุปผลการทดสอบ

5.1.1 PC/104 รุ่น PCM-3341 + PCM-3643

PCM-3341 + PCM-3643 ที่ใช้ลินุกซ์ที่ผู้ทดสอบคอมไพล์เองเป็นระบบปฏิบัติการนั้นทำงานได้อย่างจำกัดเพราะว่าช่องทางข้อมูลอนุกรมของ PCM-3341 + PCM-3643 นี้อยู่บนบัสแบบไอเอสเอซึ่งการจะใช้ไออาร์คิวร่วมได้จะต้องมีโปรแกรมแกรมขับที่เหมาะสมกับฮาร์ดแวร์ของช่องทางข้อมูลอนุกรมส่วนต่อขยายที่เกินกว่า 4 ช่องมาตรฐาน และเนื่องจากทางผู้ทดสอบไม่สามารถหาโปรแกรมขับที่เหมาะสมกับลินุกซ์ได้ทำให้ PCM-3341 + PCM-3643 ไม่สามารถรับส่งข้อมูลผ่านช่องทางข้อมูล

อนุกรมทั้ง 8 ช่องได้พร้อมกัน การทดสอบจึงเหลือเพียง 4 ช่องทางมาตรฐานเท่านั้น ซึ่งผลการทดสอบบน PCM-3341 ที่แต่ละช่องทางใช้ไออาร์คิวต่างกันสามารถทำงานได้สูงสุดถึงอัตราบอด 115200 บิตต่อวินาที โดยที่อัตราบอด 9600 บิตต่อวินาที ในสภาวะภาระงานปกติด้วยกรรมวิธีรับส่งสัญญาณจะมี excess time ในการรับส่ง 50.4 และ 26.1 มิลลิวินาที ที่การแบ่งเวลา 10 และ 100 มิลลิวินาที ตามลำดับ แสดงให้เห็นว่าเมื่อใช้กรรมวิธีรับส่งสัญญาณแล้วการแบ่งเวลาที่สั้นลงไม่ได้ช่วยปรับปรุงให้ระบบดีขึ้นในทางตรงข้ามระบบนั้นอาจทำงานได้แย่ลง ส่วนการทดสอบกับ PCM-3643 ที่เป็นส่วนต่อขยายช่องทางข้อมูลอนุกรมที่ทุกช่องทางใช้ไออาร์คิวร่วมกัน ระบบสามารถทำงานได้ถึงอัตราบอด 300 ~ 600 บิตต่อวินาทีเท่านั้นทั้งกรรมวิธีรับส่งสัญญาณและกรรมวิธีการหยั่งสัญญาณที่การแบ่งเวลา 10 และ 100 มิลลิวินาที จึงไม่สามารถนำ PCM-3341 + PCM-3643 มาใช้งานได้ตามเป้าหมาย

5.1.2 RISC base embedded computer รุ่น UC-7408

UC-7408 สามารถรับส่งไบต์ข้อมูลผ่านช่องทางข้อมูลอนุกรมได้พร้อมกันทั้ง 8 ช่องทางที่อัตราบอด 9600 บิตต่อวินาทีในสภาวะภาระงานปกติด้วยกรรมวิธีรับส่งสัญญาณจะมี excess time สูงสุด 7.3 มิลลิวินาที เมื่อมีปริมาณงาน 70440 บิตต่อวินาที ซึ่งคิดเป็นประมาณ 10 % ของเวลาที่ใช้รับส่งกรอบความยาว 70 ไบต์ ส่วนในสภาวะภาระงานสูงสุดซึ่งระบบจะพยายามส่งข้อมูลจนเต็มบัฟเฟอร์ด้านส่งอยู่ตลอดเวลานั้นเมื่อใช้วิธีรับส่งสัญญาณการกระจายของ excess time อยู่ที่ 362.726 มิลลิวินาที ซึ่งค่อนข้างมากเนื่องจากการใช้บัฟเฟอร์ขาส่งขนาดใหญ่เกินจำเป็นถึง 4096 ไบต์ ส่วนขีดจำกัดของระบบในกรรมวิธีการหยั่งสัญญาณนั้นระบบทำงานได้แค่อัตราบอด 9600 บิตต่อวินาทีเท่านั้นในขณะที่กรรมวีธีรับส่งสัญญาณระบบทำงานได้ถึงอัตราบอด 115200 บิตต่อวินาที ซึ่งจากผลดังกล่าวสรุปได้ว่าสามารถนำ UC-7408 มาทำเป็นตัวควบคุมการสื่อสารบนเครื่องบินได้

5.2 ข้อเสนอแนะ

การใช้ไออาร์คิวร่วมในพีซี/104 ที่มีบัสแบบไอเอสเอนั้นมีความจำเป็นอย่างยิ่งที่จะต้องมีโปรแกรมขับที่เหมาะสม แต่ถ้าหากไม่สามารถหาโปรแกรมขับได้เหมือนกับการทดสอบในวิทยานิพนธ์ฉบับนี้ก็มีความจำเป็นจะต้องเขียนโปรแกรมขับขึ้นเองแต่ทว่าการจะเขียนโปรแกรมขับได้ผู้เขียนจะต้องมีข้อมูลในส่วนฮาร์ดแวร์อย่างละเอียดพอสมควร และเมื่อดูจากอุปกรณ์ที่ใช้ทดสอบทั้งสองแบบจะเห็นได้ว่าการเลือกใช้อุปกรณ์ที่เหมาะสมมาทำงานตั้งแต่แรกมีความสำคัญมากพอสมควรเพราะปัจจุบันมีอุปกรณ์ออกมาให้เลือกใช้อยู่มากมายทั้งทางฮาร์ดแวร์และซอฟต์แวร์เช่นพีซี/104 ที่ใช้ทดสอบนี้ผู้ใช้จะต้องหาซอฟต์แวร์เองในขณะเดียวกัน RISC คอมพิวเตอร์ที่ทดสอบเป็นอุปกรณ์กึ่งสำเร็จรูปที่ฮาร์ดแวร์และซอฟต์แวร์มาพร้อมกันและดูเหมือนว่า RISC คอมพิวเตอร์ที่ทดสอบจะออกแบบมาตรงกับงานที่ต้องการคือเป็นตัวควบคุมการสื่อสารและจากผลการทดสอบก็ปรากฏว่า RISC คอมพิวเตอร์

สามารถทำงานได้ดี แต่ถ้าหากไม่ต้องการจะดูที่ตัวผลิตภัณฑ์โดยตรงในงานที่ต้องเชื่อมต่อกับอุปกรณ์หลาย ๆ ตัวควรเลือกใช้ฮาร์ดแวร์ที่ใช้บัสแบบพีซีไอจะดีกว่าแผงที่ใช้บัสแบบไอเอสเอเพราะบัสแบบพีซีไอทำงานโดยใช้ไออาร์คิวร่วมได้ดีกว่าบัสแบบไอเอสเอ แต่อย่างไรก็ตามการจะเลือกใช้อุปกรณ์ได้อย่างเหมาะสมนั้นผู้เลือกก็ต้องมีความรู้ในตัวผลิตภัณฑ์ที่มีวางจำหน่ายอยู่มากพอควรจึงจะเลือกได้อย่างถูกต้อง

รายการอ้างอิง

(1) Embedded system[Online]. (n.d.). Available from: http://en.wikipedia.org/wiki/Embedded_system[2007, April 1]

(2) Linux-2.x.x.x.tar[Computer file]. (n.d.). Available from: http://www.kernel.org[2007, April 1]

(3) Daniel, P., Bovet., and Marco, Cesati. Understanding the linux kernel. 1st ed. CA: O'Reilly & Associates, 2001.

(4) Alessandro, Rubini., and Jonathan, Corbet. Linux device drivers. 2nd ed. CA: O'Reilly & Associates, 2001.

(5) YoLinux Tutorial: POSIX thread (pthread) libraries[Online]. (n.d.). Available from: http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html[2007, April 1]

(6) David, S., Lawyer. Serial HOWTO[Online]. 2007. Available from: http://tldp.org/HOWTO/Serial-HOWTO.html[2007, April 1]

(7) Michael, R., Sweet. Serial Programming Guide for POSIX Operating Systems[Online]. 2005. Available from: http://www.easysw.com/~mike/serial/serial.html[2007, April 1]

(8) Cyclic redundancy check[Online]. (n.d.). Available from: http://en.wikipedia.org/wiki/Cyclic_redundancy_check[2007, April 1]

(9) ศัพท์คอมพิวเตอร์และเทคโนโลยีสารสนเทศ ฉบับราชบัณฑิตยสถาน. พิมพ์ครั้งที่ 7. กรุงเทพฯ: นานมีบุ๊คส์พับลิเคชั่นส์, 2549.

ภาคผนวก

ภาษาอังกฤษ	ภาษาไทย	ที่มา *
Address space	ปริภูมิเลขที่อยู่	2
Assembly	แอสเซมบลี	1
Attribute	แอ็ตทริบิว	2
Bluetooth	บลูทูท	1
Built-in	ในตัว	1
Cancellation point	จุดยกเลิก	2
Canonical	คานอนนิแคล	2
Cascade	แบบต่อเรียง	1
Codeword	คำรหัส	2
Command line	คำสั่งงานเชิงบรรทัด	1
Compile	คอมไพล์	3
Condition variable	ตัวแปรเงื่อนไข	2
Crossover	ครอสโอเวอร์	2
Dataflow	กระแสข้อมูล	1
Detach	ดีแทช	2
Dip	ดิป	2
Endian	เอ็นเดียน	2
Flash	แฟลช	1
Flow rate	อัตราสายงาน	2
Generator polynomial	พหุนามก่อกำเนิด	2
Group ID	ไอดีกลุ่ม	2
Handler function	ฟังก์ชันจัดกระทำ	2
Inode	ไอโนด	2
Intel	อินเทล	2
Interrupt Service Routine	รูทีนบริการสัญญาณขัดจังหวะ	2
Kernel	เคอร์เนล	3
Key polynomial	พหุนามกุญแจ	2
Legacy	เลกาซี	2
Mark	มาร์ค	2
Microcontroller	เครื่องควบคุมระดับไมโคร	2

ภาษาอังกฤษ	ภาษาไทย	ที่มา *
Mutex	มิวเท็กซ์	2
Patch	ปะ, การปะ, ตัวปะ	2
Pending signal	สัญญาณค้าง, สัญญาณคอย	2
Pentium	เพนเทียม	2
POSIX	โพสิกซ์	2
Preemptive	พรีเอมทีพ	2
Process ID	ไอดีกระบวนการ	2
Race condition	เงื่อนไขแข่งขัน	2
Raw	ดิบ	1
Reentrant	กลับเข้าใหม่	2
Return	รีเทิร์น	2
Reverse	ผันกลับ	1
Semaphore	เซมาฟอร์	2
Signal-handler function	ฟังก์ชันจัดกระทำสัญญาณ	2
Simulate	จำลอง	1
Slave	ลูกข่าย	1
Stack	สแต็ก	2
Switch	สลับ, สวิตช์	1, 2
Tarball	ทาร์บอล	2
Termination	การเลิก	2
UART	ยูอาร์ต	2
User ID	ไอดีผู้ใช้	2
Utility	อรรถประโยชน์	1
Watchdog	วอตช์ดอก	2

ที่มา * 1. ผู้เขียนตัดต่อมาจาก (9)

2. ผู้เขียนทำการเขียนขึ้นมาเองโดยพยายามใช้การทับศัพท์เพราะไม่มีบัญญัติใน (9)

3. ผู้เขียนใช้การทับศัพท์แม้ว่าจะมีบัญญัติใน (9) แล้วแต่ผู้เขียนเห็นว่าหากใช้คำศัพท์ตามที่บัญญัติใน (9) อาจทำให้เกิดความลำบากในการอ่านจึงขออนุญาตเขียนทับศัพท์

ส่วนคำศัพท์อื่นที่ไม่ได้แสดงในภาคผนวกนี้คือคำศัพท์ที่ผู้เขียนใช้ตามที่ได้บัญญัติไว้ใน (9)

ประวัติผู้เขียนวิทยานิพนธ์

นายนครินทร์ เหล่าวัฒนาถาวร เกิดเมื่อวันที่ 26 พฤศจิกายน พ.ศ.2522 สำเร็จปริญญาวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมไฟฟ้า จากจุฬาลงกรณ์มหาวิทยาลัย ปีการศึกษา 2544