การจัดตารางงานในระบบปฏิบัติการแบบเวลาจริงบนไมโครโพรเซสเซอร์ชนิดหลายแกน

นายอังคาร เชี่ยวกิจวุฒิกุล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2555

SCHEDULING TASKS IN REAL-TIME OPERATING SYSTEMS ON MULTIPLE CORE MICROPROCESSORS

Mr. Angkhan Chiewkijwutthikul

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Software Engineer
Department of Computer Engineering
Faculty of Engineer
Chulalongkorn University
Academic Year 2012

หัวข้อวิทยานิพนธ์	การจัดตารางงานในระบบปฏิบัติการแบบเวลาจริงบนไมโครโพรเซสเซอร์ชนิดหลายแกน
โดย	นายอังคาร เชี่ยวกิจวุฒิกุล
สาขาวิชา	วิศวกรรมซอฟต์แวร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ศาสตราจารย์ ดร.ประภาส จงสถิตย์วัฒนา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ
(อาจารย์ ดร.ยรรยง เต็งอำนวย)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(ศาสตราจารย์ ดร.ประภาส จงสถิตย์วัฒนา)

.....กรรมการภายนอกมหาวิทยาลัย
(รองศาสตราจารย์ ดร.วรา วราวิทย์)

อังคาร เชี่ยวกิจวุฒิกุล : การจัดตารางงานในระบบปฏิบัติการแบบเวลาจริงบนไมโครโพรเซสเซอร์ชนิดหลายแกน. (SCHEDULING TASKS IN REAL-TIME OPERATING SYSTEMS ON MULTIPLE CORE MICROPROCESSORS) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ศ.ดร.ประภาส จงสถิตย์วัฒนา, 66 หน้า.

งานวิจัยนี้นำเสนอวิธีการจัดตารางงานในระบบปฏิบัติการแบบเวลาจริงบนหน่วยประมวลผลชนิดหลายแกน เพื่อให้ระบบปฏิบัติการแบบเวลาจริงใช้งานทรัพยากรของหน่วยประมวลผลที่มีจำนวนแกนที่เพิ่มขึ้นอย่างคุ้มค่า โดยพัฒนาต้นแบบจากระบบปฏิบัติการ Micrium μC/OS-III ให้สามารถทำงานบนหน่วยประมวลผล S2 เป็นหน่วยประมวลผลชนิดสองแกน ใช้การจัดตารางงานแบบการปรับดุลจำนวนส่วนงานระหว่างแกนหน่วยประมวลผลโดยให้ส่งงานไปยังแกนหน่วยประมวลผลที่มีภาระงานน้อยกว่าก่อน ทำการทดสอบเปรียบเทียบประสิทธิภาพการทำงานของระบบบนหน่วยประมวลผลชนิดแกนเดียวกับหน่วยประมวลผลชนิดสองแกน ผลการวิจัยพบว่าการทำงานของระบบปฏิบัติการบนหน่วยประมวลผลชนิดสองแกนมีประสิทธิภาพเพิ่มขึ้น 171 เปอร์เซ็นต์ จากการทำงานบนหน่วยประมวลผลแกนเดียว

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมซอฟต์แวร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2555

5471034821 : MAJOR SOFTWARE ENGINEERING
KEYWORDS : TASK SCHEDULING / OPERATING SYSTEMS / MULTICORE COMPUTING

ANGKHAN CHIEWKIJWUTTHIKUL : SCHEDULING TASKS IN REAL-TIME OPERATING SYSTEMS ON MULTIPLE CORE MICROPROCESSORS. ADVISOR : PROF. PRABHAS CHONGSTITVATANA, Ph.D., 66 pp.

This work proposed a scheduling scheme of a real-time operating system for multicore processors. The aim is to use resources efficiently. The development is based on Micrium µC/OS-III. The program is ported to run on S2 dual core processor. The scheduler adjusts the work balance between two cores. The work is sent to the core that has less work first. The experiment is carried out to compare the performance of running tasks on single core and multiple core microprocessors. The result shows that the efficiency is increased by 171 percent on a dual core processor compare to a single core processor.

Department : Computer Engineering Student's Signature .....
Field of Study : Software Engineering Advisor's Signature .....
Academic Year : 2012 .....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จสมบูรณ์ได้เนื่องจากได้รับความกรุณาจากศาสตราจารย์ ดร. ประภาส จงสถิตย์วัฒนา รับเป็นที่ปรึกษาและให้คำแนะนำ จนทำให้วิทยานิพนธ์ฉบับนี้เสร็จสิ้นสมบูรณ์ อีกทั้งขอขอบพระคุณอาจารย์ ดร.ยรรยง เต็งอำนวย และรองศาสตราจารย์ ดร.วรา วราวิทย์ ซึ่งเป็นประธานกรรมการและกรรมการ ที่ได้เสียสละเวลา ให้คำชี้แนะ ชี้ข้อบกพร่องรวมถึงแนะนำแนวทางการวิจัยให้ข้าพเจ้า

สุดท้ายนี้ขอกราบขอบพระคุณคุณแม่ ขอขอบคุณพี่ชายทุกคน พี่จุลเทพ พี่ไพโรจ และน้องเศรษฐ์จิรภา ที่ให้คำปรึกษาและคอยเป็นกำลังใจเสมอมา

สารบัญ

หน้า

สารบัญตาราง

หน้า

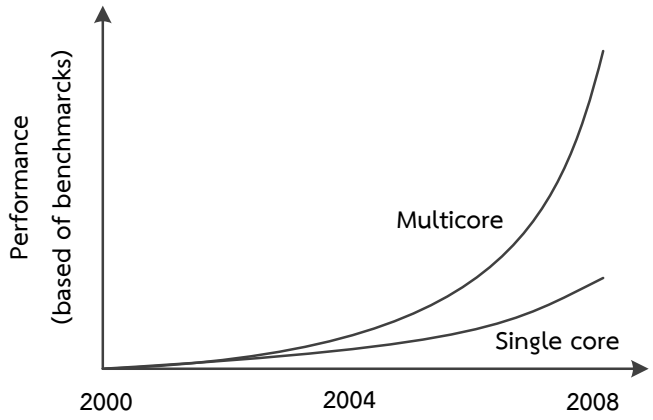
สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ระบบปฏิบัติการแบบเวลาจริง (Real-Time Operating System) เป็นระบบที่การทำงานและการตอบสนองการทำงานหรือการใช้เวลาเป็นตัววัดประสิทธิภาพของระบบ โดยทั่วไประบบปฏิบัติการแบบเวลาจริงถูกนำไปใช้ในซอฟต์แวร์แบบระบบฝังตัว (Embedded System) เป็นระบบที่ทำขึ้นเฉพาะทาง ซึ่งได้ถูกนำไปใช้อย่างแพร่หลาย ในหลายอุตสาหกรรม อุตสาหกรรมยานยนต์ เครื่องควบคุมต่างๆ เครื่องใช้ไฟฟ้าทั้งภายในบ้านและภายในสำนักงาน โดยเน้นไปในอุปกรณ์ที่ต้องการควบคุมการตอบสนองการทำงานในเวลาที่ต้องการ

ระบบฝังตัวจะทำงานบนฮาร์ดแวร์ที่แตกต่างกันไปตามความเหมาะสมของงาน โดยมีหน่วยประมวลผลเป็นหัวใจ เพราะเป็นส่วนสำคัญที่มีผลต่อประสิทธิภาพของระบบ โดยแต่ละหน่วยประมวลผลจะมีสถาปัตยกรรมและความเร็วในการทำงานแตกต่างกัน หน่วยประมวลผลที่มีความเร็วในการทำงานที่มีสูงกว่ามักจะทำให้ระบบมีประสิทธิภาพสูงกว่า หน่วยประมวลผลได้มีการพัฒนา โดยเพิ่มแกนหน่วยประมวลผลจากแกนเดียว (Single Core Microprocessor) เป็นหน่วยประมวลผลหลายแกน (Multiple Core Microprocessor)



ภาพที่ 1.1 แสดงแนวโน้มประสิทธิภาพการทำงานระหว่างหน่วยประมวลผลหลายแกนและแกนเดียว [1]

ซึ่งในปัจจุบันระบบฝังตัวมีแนวโน้มการใช้หน่วยประมวลผลหลายแกนมากขึ้น [1] เพราะหน่วยประมวลผลหลายแกนสามารถเพิ่มประสิทธิภาพการทำงานของระบบให้ดียิ่งขึ้น เมื่อเปรียบเทียบกับการทำงานบนหน่วยประมวลผลแกนเดียว ดังภาพที่ 1.1 แสดงประสิทธิภาพการทำงานของหน่วยประมวลผลหลายแกนที่เพิ่มขึ้นมากกว่าหน่วยประมวลผลแกนเดียว

เนื่องจากแกนของหน่วยประมวลผลมีจำนวนเพิ่มขึ้น ระบบปฏิบัติการแบบเวลาจริงที่มีการจัดตารางงานทำหน้าที่เลือกส่วนงานเพื่อทำงานซึ่งเป็นเครื่องจักรสำคัญในการทำงานและส่งผล

กระทบต่อประสิทธิภาพของระบบ ดังนั้นจึงมีความจำเป็นที่ต้องมีการพัฒนาการจัดตารางงานในระบบปฏิบัติการแบบเวลาจริง ให้สามารถใช้งานบนหน่วยประมวลผลหลายแกนเช่นกัน เพื่อให้ระบบสามารถใช้แกนหน่วยประมวลผลที่มีจำนวนเพิ่มขึ้นอย่างมีประสิทธิภาพ

งานวิจัยนี้นำเสนอวิธีการจัดตารางงานในระบบปฏิบัติการแบบเวลาจริงบนหน่วยประมวลผลหลายแกน โดยพัฒนาการจัดตารางงานบนระบบปฏิบัติการแบบเวลาจริงให้สามารถทำงานบนหน่วยประมวลผลชนิดหลายแกนเพื่อให้ใช้งานทรัพยากรอย่างคุ้มค่า

1.2 วัตถุประสงค์ของการวิจัย

เสนอการจัดตารางงานในระบบปฏิบัติการแบบเวลาจริงบนหน่วยประมวลผลหลายแกนเพื่อให้ใช้ทรัพยากรอย่างคุ้มค่า

1.3 ขอบเขตของการวิจัย

พัฒนาระบบจากระบบปฏิบัติการแบบเวลาจริงต้นแบบ Micrium µC/OS-III [2] และพัฒนาบนหน่วยประมวลผล S2 : A Hypothetical 32-bit Processor Version 3 [3] เป็นหน่วยประมวลผลชนิดสองแกน ที่ถูกจำลองขึ้นเพื่อใช้ในการศึกษา โดยมีลักษณะเป็นหน่วยประมวลผลชนิดหลายแกนแบบสมมาตร ใช้ภาษา Rz และภาษา Assembly ในการพัฒนา

1.4 ประโยชน์ที่คาดว่าจะได้รับ

ได้วิธีการต้นแบบที่ใช้ในการจัดตารางงานในระบบปฏิบัติการแบบเวลาจริงบนหน่วยประมวลผลหลายแกนและระบบปฏิบัติการแบบเวลาจริงที่สามารถทำงานบนหน่วยประมวลผลหลายแกน

1.5 วิธีดำเนินการวิจัย

1) ศึกษาระบบปฏิบัติการ Micrium µC/OS-III ให้เข้าใจถึงโครงสร้างการทำงานส่วนประกอบต่าง ๆ ของระบบปฏิบัติการ การจัดตารางงานและเครื่องมือต่าง ๆ ที่มีในระบบปฏิบัติการ

2) ศึกษาหน่วยประมวลผล S2 ให้เข้าใจการทำงานและศึกษาการใช้งาน

3) ออกแบบการจัดตารางงาน

4) ทำการย้ายระบบปฏิบัติการ Micrium µC/OS-III ลงบนหน่วยประมวลผล โดยทำการเปลี่ยนจากภาษา C เป็นภาษา Rz เพื่อใช้งานบนหน่วยประมวลผล S2

5) พัฒนาส่วนจัดตารางงาน

6) ทดสอบและวัดประเมินผล

7) จัดทำสรุปผลงานวิจัยและข้อเสนอแนะของงานวิจัย

8) จัดทำวิทยานิพนธ์

1.6 ผลงานตีพิมพ์จากวิทยานิพนธ์

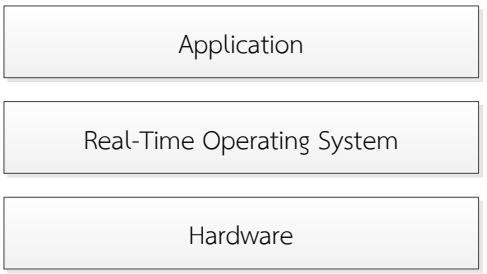
ส่วนหนึ่งของงานวิทยานิพนธ์ได้รับการตีพิมพ์เป็นบทความวิชาการในหัวเรื่อง “Scheduling Tasks in Real-time Operating Systems on Multiple Core Microprocessors” โดยอังคาร เชี่ยวกิจวุฒิกุล และประภาส จงสถิตย์วัฒนา ในบันทึกการประชุม “The 9th National Conference on Computing and Information Technology, NCCIT2013” ซึ่งจัดขึ้น ณ คณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ (KMUTNB) ประเทศไทย ระหว่างวันที่ 9-10 พฤษภาคม 2556

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้องกับงานวิจัย

2.1.1 ระบบปฏิบัติการแบบเวลาจริง

ระบบปฏิบัติการแบบเวลาจริง (Real-Time Operating System) เป็นระบบปฏิบัติการที่ออกแบบมาเพื่อตอบสนองความต้องการ การทำงานในลักษณะเวลาจริงหรือตอบสนองการทำภารกิจตามเวลาที่กำหนด โดยตัวระบบปฏิบัติการจะทำหน้าที่เป็นตัวกลางการจัดการระหว่างทรัพยากรบนฮาร์ดแวร์และตัวโปรแกรมประยุกต์ ดังภาพที่ 2.1



ภาพที่ 2.1 ระบบปฏิบัติการแบบเวลาจริง

ระบบปฏิบัติการแบบเวลาจริงทั่วไปจะสามารถเพิ่มหรือลดความสามารถให้ตรงตามความต้องการของระบบได้ เพื่อให้เกิดความเหมาะสมกับตัวฮาร์ดแวร์และทำให้ระบบมีความคุ้มค่าที่สุด

2.1.2 การจัดตารางงาน

ระบบเวลาจริงบนหลายหน่วยประมวลผล (Real-time Multiprocessor System) สามารถแบ่งจำแนก [4,5] ได้ตาม

2.1.2.1 การแบ่งสรร (Allocation)

1) แบบไม่มีการโยกย้าย (No Migration) คือการจัดตารางการทำงานแบบแบ่งส่วน (Partitioned Scheduling Algorithms) โดยแต่ละส่วนงานทำการจองการทำงานลงหน่วยประมวลผลหน่วยหนึ่งและไม่มีการโยกย้ายไปยังหน่วยประมวลผลอื่น

2) แบบการโยกย้ายระดับส่วนงาน (Task-level Migration) คือส่วนงานอาจสามารถทำงานบนหน่วยประมวลผลที่ต่างกัน แต่เมื่อทำงานแล้วจะทำอยู่บนหน่วยประมวลผลนั้นๆ จนจะมีการเปลี่ยนจากตัวจัดการอีกครั้ง

3) แบบการโยกย้ายระดับส่วนชิ้นงาน (Job-level Migration) คือส่วนงานสามารถทำงานบนหน่วยประมวลผลที่ต่างกันได้ แต่ยังไม่สามารถทำงานแบบขนานหรือพร้อมกัน สามารถโยกย้ายโดยไม่มีข้อจำกัดว่าต้องรอให้เสร็จงานนั้น ๆ ก่อนขึ้นกับตัวจัดตารางงาน

2.1.2.2 การให้ลำดับความสำคัญ (Priority)

1) การลำดับความสำคัญของส่วนงานแบบคงที่ (Fixed Task Priority) คือแต่ละงานจะมีการให้ระดับความสำคัญค่าหนึ่งและเป็นค่าให้กับทุกส่วนชิ้นงาน

2) การลำดับความสำคัญของส่วนชิ้นงานแบบคงที่ (Fixed Job Priority) คือแต่ละส่วนงานสามารถมีค่าระดับความสำคัญต่างกันแต่ส่วนชิ้นงานจะมีค่าเดียวกัน

3) การลำดับความสำคัญแบบพลวัต (Dynamic Priority) คือแต่ละส่วนชิ้นงานสามารถมีค่าระดับความสำคัญต่างกันไปในแต่ละช่วงเวลา

2.1.2.3 การแบ่งตามการอนุรักษ์งาน (Work-Conserving)

1) แบบให้สิทธิ์ก่อน (Pre-Emptive) คือการให้ส่วนงานที่มีระดับความสำคัญสูงสุดได้สิทธิ์ทำงานก่อนเสมอ

2) แบบไม่ให้สิทธิ์ก่อน (Non-Pre-Emptive) คือเมื่อส่วนงานหนึ่งกำลังทำงานจะไม่สามารถเรียกส่วนงานอื่นทำงานจนกระทั้งทำงานเสร็จ

3) แบบทำงานร่วมกัน (Co-operative) คือส่วนงานสามารถทำงานพร้อมกันสลับกันไปโดยไม่มีการให้ระดับความสำคัญกับส่วนงาน

2.1.3 การตรวจวัดประสิทธิภาพจัดตารางงาน

สามารถอธิบายการตรวจวัดประสิทธิภาพการจัดตารางงานในระบบปฏิบัติการแบบเวลาจริงออกเป็น 4 แบบ [4]

1) ขอบเขตการใช้ประโยชน์ (Utilization Bound)

เป็นวิธีวัดใช้สำหรับส่วนงานแบบรู้กรอบเวลาการทำงานแน่นอน (Implicit-Deadline Task Sets) โดยวัดจากการใช้ประโยชน์กรอบเวลางานที่มี

2) อัตราส่วนประมาณ (Approximation Ratio)

เป็นการเปรียบเทียบประสิทธิภาพของการจัดตารางงานต้นแบบเทียบกับตัวจัดการตารางงานที่ดีที่สุด พิจารณาการกำหนดจำนวนขั้นต่ำของหน่วยประมวลผลที่ใช้ในการทำตารางงาน

3) การเพิ่มทรัพยากร (Resource Augmentation or Speedup Factors)

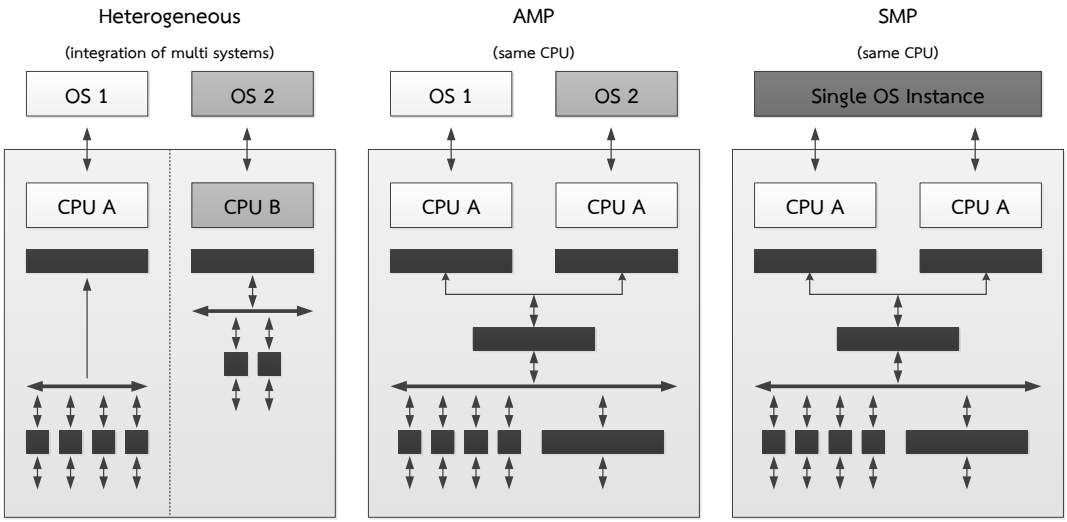
เป็นการเปรียบเทียบประสิทธิภาพของการจัดตารางงานต้นแบบเทียบกับตัวจัดการตารางงานที่ดีที่สุด พิจารณาเวลาการทำงานของหน่วยประมวลผลที่ใช้ในการทำงานตารางงาน

4) ทำการทดลอง (Empirical Measures)

ทำการวัดเปรียบเทียบจำนวนส่วนงานที่สร้างขึ้นโดยเปรียบเทียบประสิทธิผลของสองวิธีการหรือมากกว่า โดยทดลองสร้างส่วนงานจากการเปลี่ยนคุณสมบัติของส่วนงานและทำการกำหนดคุณสมบัติคงที่ให้กับส่วนงาน แล้วดูความเป็นไปได้ในการจัดตารางงาน

2.1.4 หน่วยประมวลผลแบบหลายแกน

หน่วยประมวลผลแบบหลายแกน [6] คือหน่วยประมวลผลที่มีการรวมหน่วยประมวลผลกลางมากกว่า 2 ตัวขึ้นไป เข้าร่วมเป็นแกนของหน่วยประมวลผลกลางจึงทำให้หน่วยประมวลผลแบบหลายแกนมีการประมวลผลมีประสิทธิภาพดีขึ้น สามารถแบ่งสถาปัตยกรรมและลักษณะการทำงานได้ดังภาพที่ 2.2



ภาพที่ 2.2 ลักษณะของหน่วยประมวลผลหลายแกนแบบต่างกัน [6]

หน่วยประมวลผลหลายแกนแบบต่างกัน (Heterogeneous) คือมีหน่วยประมวลผลกลางที่นำมาใช้เป็นแกนมีลักษณะไม่เหมือนกัน หรือมีการแบ่งหน่วยประมวลผลออกเป็นตัวหลักและตัวรอง โดยให้ตัวหลักทำงานส่วนใหญ่และตัวรองทำงานเฉพาะทางตามลักษณะงาน เช่นหน่วยประมวลผลที่มีหน่วยประมวลผลตัวรองไว้ทำการเข้ารหัสเสียงอย่างเดียวและให้ตัวหลักทำงานอื่นๆ

หน่วยประมวลผลหลายแกนแบบเหมือนกัน (Homogeneous) คือมีหน่วยประมวลผลกลางที่ถูกใช้เป็นแกนมีลักษณะเหมือนกัน โดยทุกหน่วยประมวลผลสามารถทำงานในลักษณะที่เหมือนกันได้

ลักษณะการทำงานของหน่วยประมวลผลหลายแกน (Multiprocessing Models) มี 2 แบบคือ

1) การประมวลผลแบบไม่สมมาตร (Asymmetric Multiprocessing: AMP) คือ การทำงานของระบบปฏิบัติการจะแยกใช้บนแกนหน่วยประมวลผล โดยจะใช้ระบบปฏิบัติการที่เหมือนกันหรือไม่เหมือนกันก็ได้

2) การประมวลผลแบบสมมาตร (Symmetric Multiprocessing: SMP) คือการทำงานโดยทุกหน่วยประมวลผลทำงานภายใต้ระบบปฏิบัติการเดียวกัน

2.1.5 ระบบปฏิบัติการ Micrium µC/OS-III

ระบบปฏิบัติการ Micrium µCOS-III [2] เป็นระบบปฏิบัติการแบบให้สิทธิ์ก่อน (Pre-emptive Real-Time Kernel) พัฒนาขึ้นปี ค.ศ. 2009 โดย ใช้ภาษา C ในการพัฒนาเป็นหลักและมีภาษา Assembly บางส่วน อธิบายลักษณะสำคัญของระบบปฏิบัติการได้ดังนี้

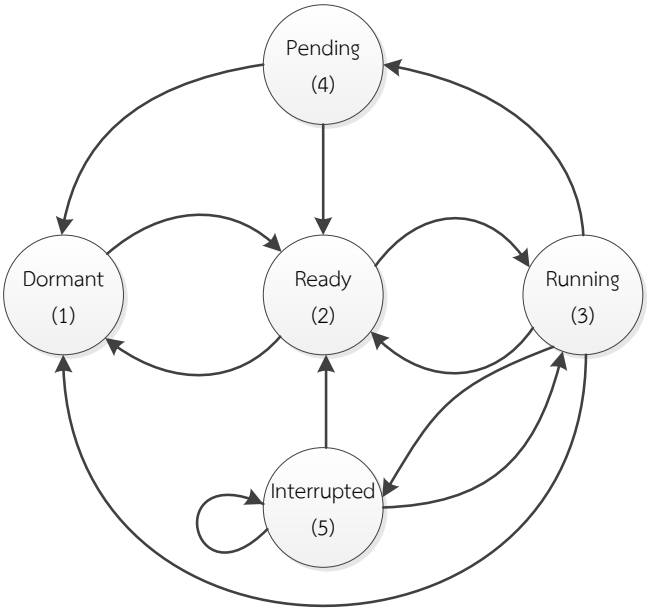
2.1.5.1 การจัดการส่วนงาน (Task Management)

สามารถทำงานได้หลายๆ งานในเวลาเดียวกัน (Multitasking) ตัวส่วนงาน (Task) หรือเทรด (Thread) มีการกำหนดระดับความสำคัญ กำหนดขนาดแสตก (Stack) การตรวจสอบขนาดแสตกและสถานะของส่วนงาน

1) การกำหนดระดับความสำคัญ (Assigning Task Priorities) ให้กับส่วนงานขึ้นกับความเหมาะสมตามโปรแกรมประยุกต์

2) การกำหนดขนาดแสตก (Determining the Size of a Stack) ให้กับส่วนงาน โดยแสตกจะถูกใช้เก็บ CPU registers และ Floating-Point Unit (FPU) registers

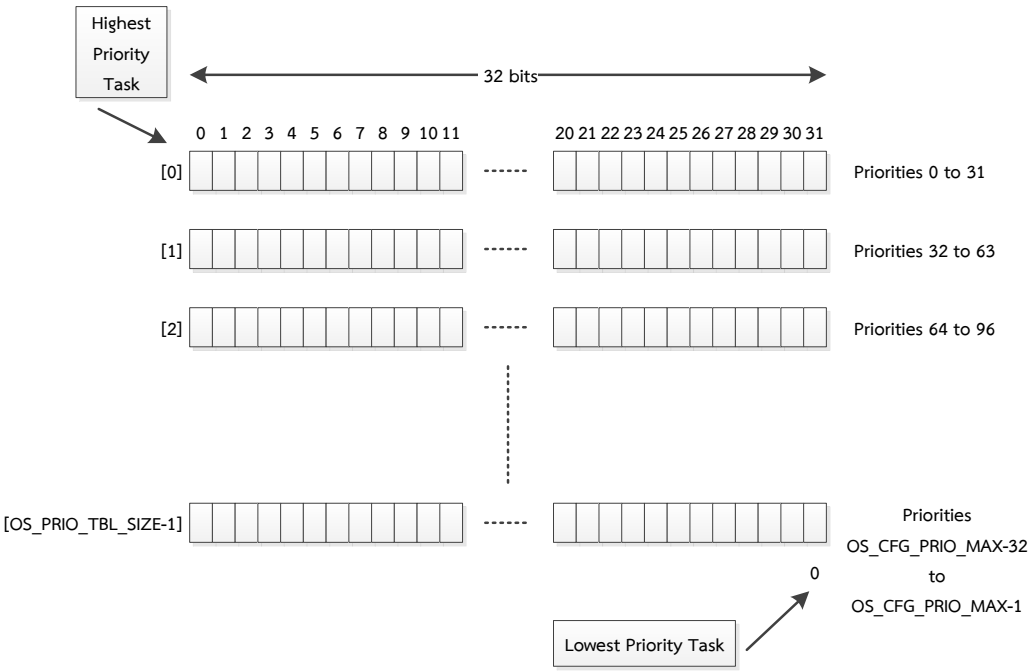
3) สถานะของส่วนงาน (Task States) มี 5 สถานะคือ 1.หลับ (Dormant) 2.พร้อม (Ready) 3.กำลังทำงาน (Running) 4.หน่วง (Pending) 5.ถูกขัดจังหวะ (Interrupted) แต่ละสถานะมีความสัมพันธ์ดังภาพที่ 2.3 โดยปกติการทำงานของส่วนงานจะเริ่มจากสถานะหลับแล้วจึงจะเปลี่ยนเป็นสถานะพร้อมทำงาน กำลังทำงานและหน่วงหรือรอการเรียกเพื่อเป็นสถานะพร้อมทำงานต่อไป โดยจะมีสถานะถูกขัดจังหวะได้ระหว่างการทำงาน



ภาพที่ 2.3 สถานะของส่วนงาน (Task States) [2]

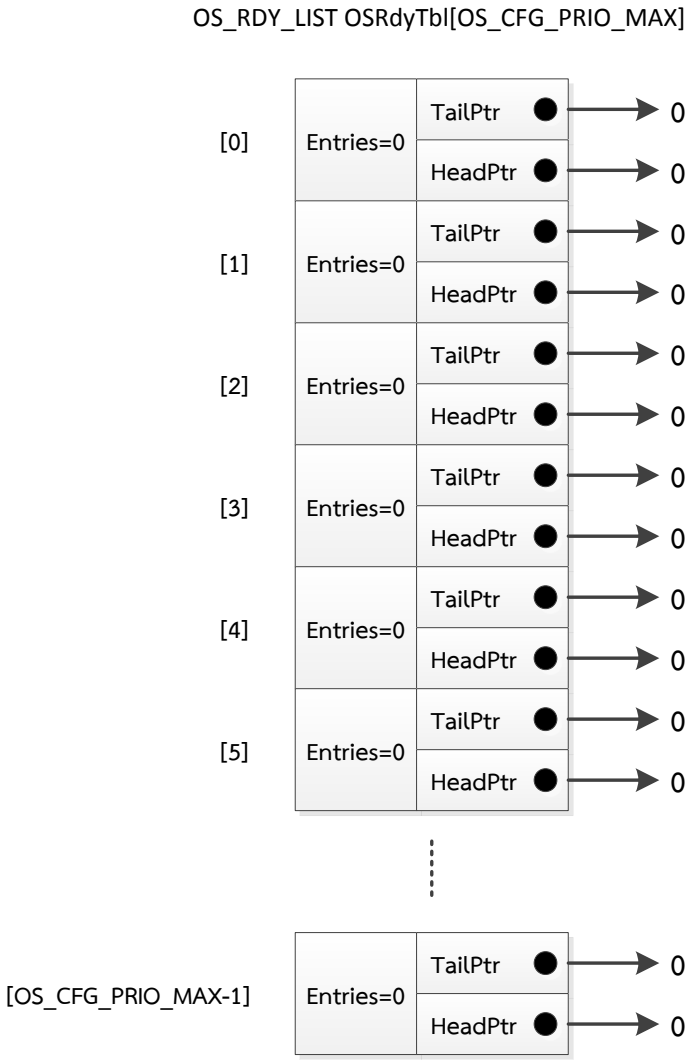
2.1.5.2 รายการสถานะพร้อมทำงาน (Ready List)

ส่วนงานที่อยู่ในสถานะพร้อมจะถูกวางใส่ในรายการสถานะพร้อมทำงาน รายการจะมีสองส่วนคือ Bitmap Priority Levels และตาราง Ready List Pointers ดังภาพที่ 2.4 และภาพที่ 2.5 ตามลำดับ



ภาพที่ 2.4 แสดง Bitmap Priority Levels [2]

ค่าลำดับความสำคัญที่มีค่าน้อยจะมีความสำคัญมากและค่าลำดับความสำคัญที่มีค่าน้อยจะมีความสำคัญมาก จากภาพที่ 2.4 เมื่อมีส่วนงานที่มีสถานะพร้อมทำงานและส่วนสัมพันธ์กับค่าลำดับความสำคัญของส่วนงานนั้นจะถูกให้ค่า (เช่นเป็น 1) ลงบน Bitmap Priority Levels เพื่อไปใช้ในการจัดตารางงานต่อไป



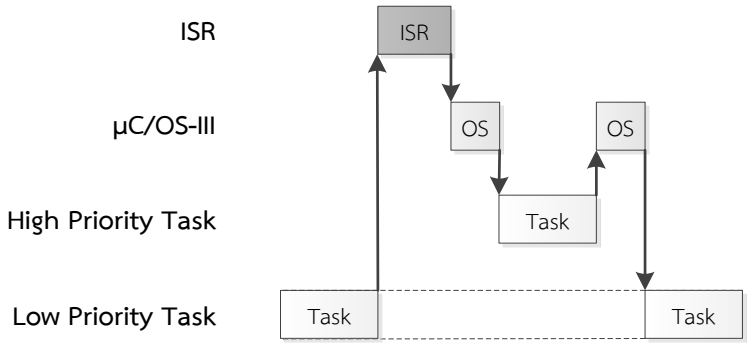
ภาพที่ 2.5 แสดงตาราง Ready List Pointers [2]

ตาราง Ready List Pointers จากภาพที่ 2.5 เป็นตารางไว้เก็บตำแหน่งชี้ไปส่วนงาน โดยมี Entries เก็บเลขจำนวนของส่วนงานที่มีสถานะพร้อมทำงานและมีความสัมพันธ์กับค่าลำดับความสำคัญบน Ready List ถ้า Entries เป็นศูนย์แสดงว่าไม่มีส่วนงานที่มีระดับความสำคัญนั้นพร้อมทำงาน TailPtr และ HeadPtr เก็บ Doubly Linked List ของส่วนงานทั้งหมดที่มีระดับความสำคัญเดียวกัน โดยที่ HeadPtr เก็บส่วนหัวและ TailPtr เก็บส่วนหาง

2.1.5.3 การจัดตารางงาน (Scheduling Algorithms)

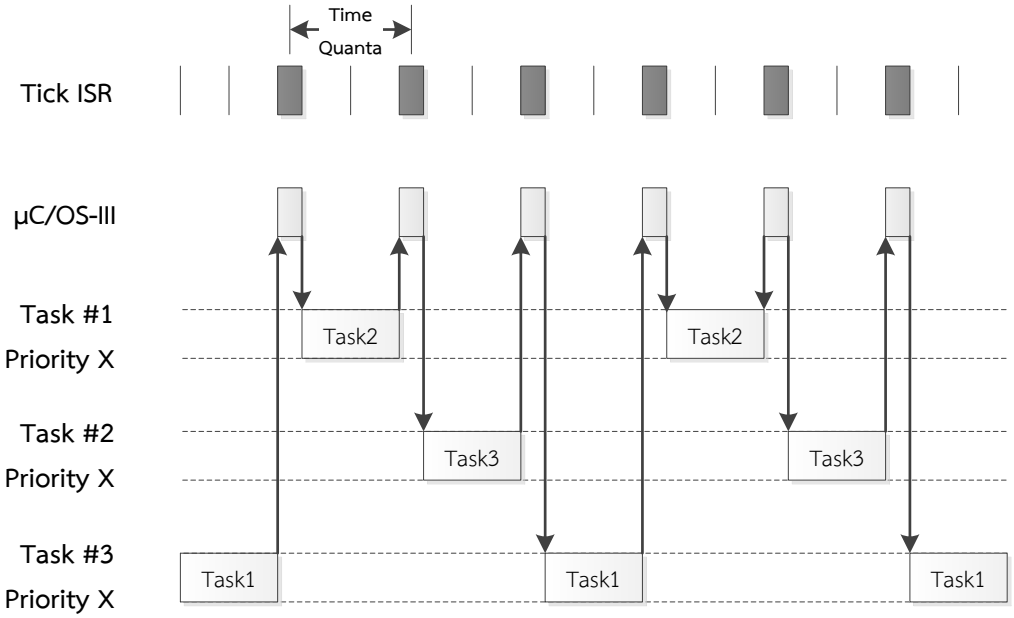
ทำหน้าที่เลือกส่วนงานเพื่อทำงานบนแกนหน่วยประมวลผลตามลำดับความสำคัญ โดยระบบปฏิบัติการ Micrium µCOS-III จะมี 2 วิธี

1) การจัดตารางงานแบบให้สิทธิ์ก่อน (Preemptive Scheduling) ทำงานในกรณีที่ระดับความสำคัญไม่เท่ากัน โดยจะให้ส่วนงานที่มีระดับความสำคัญทำงานก่อนเสมอ ดังภาพที่ 2.6



ภาพที่ 2.6 การจัดตารางงานแบบให้สิทธิ์ก่อน [2]

2) การจัดตารางงานแบบวนรอบ (Round Robin Scheduling) ทำงานในกรณีที่ระดับความสำคัญของแต่ละส่วนงานเท่ากัน โดยทำการแบ่งช่วงเวลาเพื่อเรียกทำงาน (Time slice) ดังภาพที่ 2.7

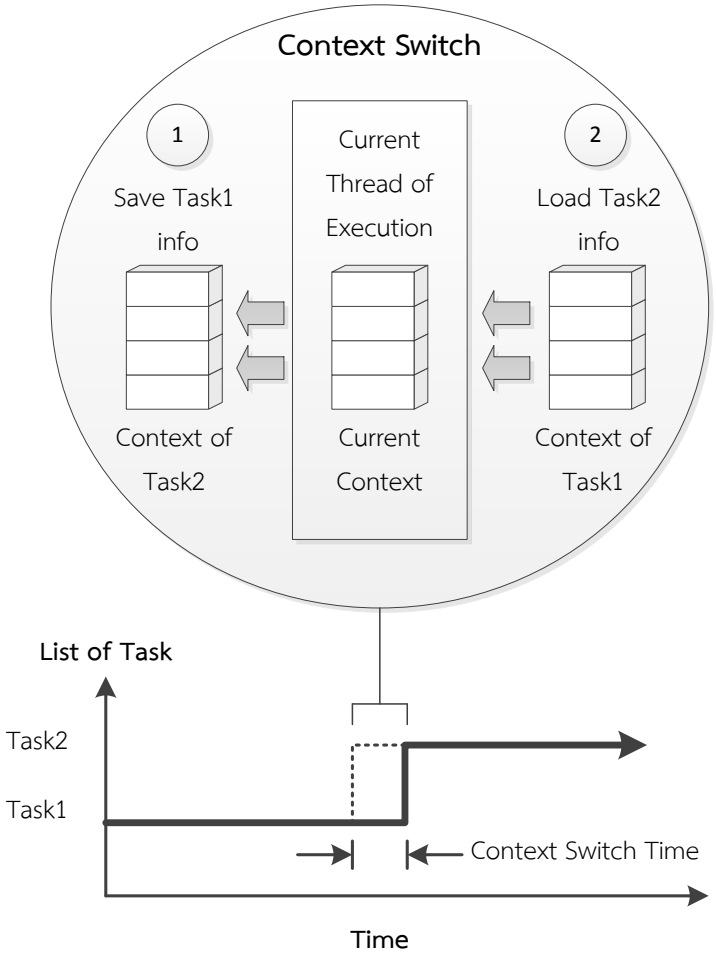


ภาพที่ 2.7 การจัดตารางงานแบบวนรอบ [2]

Time Quanta สามารถปรับเปลี่ยนจากส่วนต่อประสานโปรแกรมประยุกต์ ตามความเหมาะสมโดยผู้พัฒนา

2.1.5.4 การเปลี่ยนส่วนงาน (Context Switching)

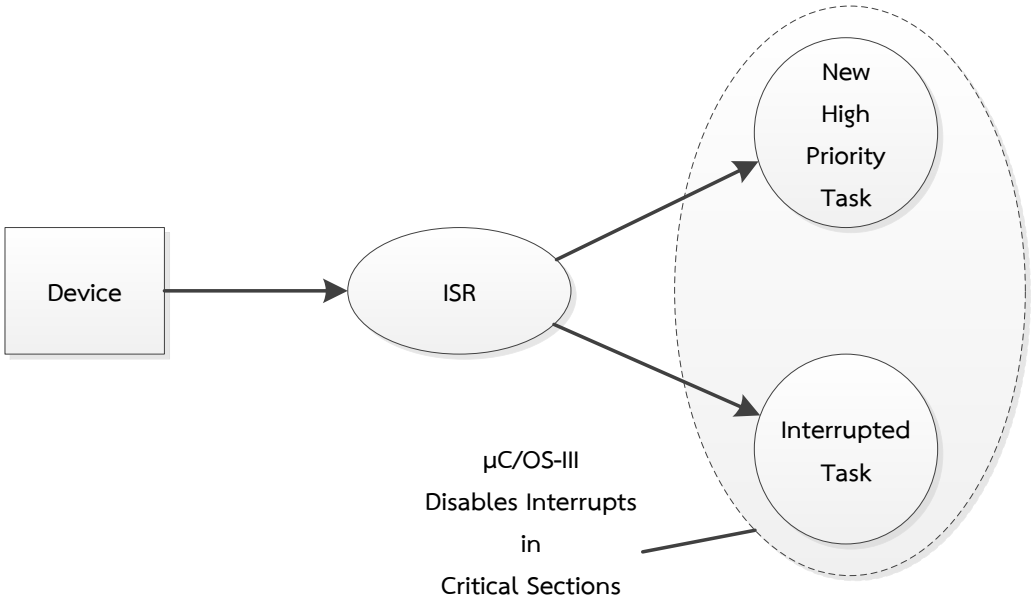
เป็นการเปลี่ยนการทำงานจากส่วนงานหนึ่งไปอีกส่วนงานหนึ่ง ดูรายละเอียดได้จากภาพที่ 2.8 แสดงการทำงานเปลี่ยนส่วนงาน โดยจะเก็บบันทึกข้อมูลปัจจุบันแล้วทำการเรียกข้อมูลส่วนงานใหม่เพื่อทำงาน



ภาพที่ 2.8 แสดงการเปลี่ยนส่วนงาน [7]

2.1.5.5 การจัดการสัญญาณขัดจังหวะ (Interrupt Management)

สัญญาณขัดจังหวะถูกใช้เรียกเหตุการณ์ต่างๆ ที่เกิดขึ้นในเวลาใดๆ เมื่อเกิดสัญญาณขัดจังหวะ จะบันทึกข้อมูลที่กำลังทำงานอยู่แล้วจะกระโดดไปยังส่วนที่เรียกว่า Interrupt Service Routing (ISR) จากนั้นจะทำส่วนงานสัญญาณขัดจังหวะ (Interrupted Task) หรือส่วนงานที่มีระดับความสำคัญสูงที่สุด ดังภาพที่ 2.9



ภาพที่ 2.9 การจัดการสัญญาณขัดจังหวะ [2]

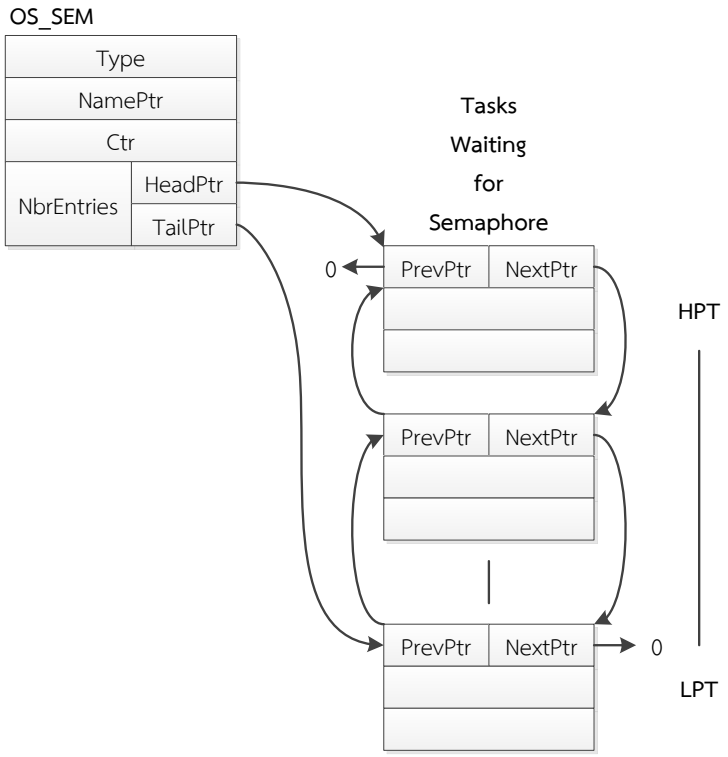
2.1.5.6 การจัดการทรัพยากร (Resource Management)

ระบบปฏิบัติการ Micrium µCOS-III มีการจัดการทรัพยากรที่ใช้ร่วมกัน เช่นตัวแปร โครงสร้างข้อมูล ตาราง (ในหน่วยความจำ) หรือ Registers ใน I/O เพื่อป้องกันการใช้ทรัพยากรร่วมกันระบบปฏิบัติการ Micrium µCOS-III จะมีตัวป้องกันดังนี้

1) การเปิดและปิดสัญญาณขัดจังหวะ (Disable/Enable Interrupts) เป็นวิธีที่ง่ายและเร็วที่สุดในการป้องกันการใช้ทรัพยากรร่วมกันโดยทำการปิดสัญญาณขัดจังหวะก่อนเรียกใช้ทรัพยากรแล้วทำการเปิดสัญญาณขัดจังหวะต่อไป

2) Lock/Unlock เป็นตัวป้องกันการเปลี่ยนส่วนงานโดยถูกใช้เมื่อส่วนงานมีการใช้ทรัพยากรร่วมกันกับ ISR ซึ่งใช้กลไกการปิดเปิดการเปลี่ยนส่วนงาน

3) Semaphores เป็นกลไกเพื่อป้องกันการใช้ทรัพยากรร่วมกันโดยมีสองแบบคือ Binary Semaphores และ Counting Semaphores มีลักษณะเป็น Signaling Mechanism โดยภาพที่ 2.10 แสดงรายการรอเรียกใช้งานของ Semaphore ซึ่งเก็บค่านับจำนวนของ Semaphore จำนวนส่วนงานรอการเรียกทำงาน HeadPtr และ TailPtr ของส่วนงานในรายการ



ภาพที่ 2.10 รายการรอเรียกใช้งานของ Semaphore [2]

4) Mutex (Mutual Exclusion Semaphores) เป็นกลไกพิเศษของ Binary Semaphore ใช้ป้องการการใช้ทรัพยากรร่วมกันระหว่างส่วนงานมีลักษณะเป็น Locking Mechanism

2.1.6 หน่วยประมวลผล S2

หน่วยประมวลผล S2 : A Hypothetical 32-bit Processor Version 3 [3] เป็นหน่วยประมวลผลชนิดสองแกน ที่ถูกจำลองขึ้นเพื่อใช้ในการศึกษาโดยเริ่มใช้ตั้งแต่ปี ค.ศ. 2001 มีลักษณะเป็นหน่วยประมวลผลชนิดสองแกนแบบเหมือนกัน มีโปรแกรมจำลองแบบการทำงาน รวมถึงการจำลองสัญญาณการขัดจังหวะและการตั้งเวลาในหน่วยประมวลผล ใช้ภาษา Rz และภาษา Assembly ในการพัฒนา

หน่วยประมวลผล S2 เป็นหน่วยประมวลผลชนิด 32 bit มี 32 registers สามารถเรียกและเก็บข้อมูลลงหน่วยความจำ มีรูปแบบเป็น Three-address instruction formats แสดงในตารางที่ 2.1

op r1 r2 r3 means R[r1] = R[r2] op R[r3]

ตารางที่ 2.1 รูปแบบ Three-address instruction formats [3]

ชนิดของ Instruction มีให้ใช้งานแบ่งออกเป็น 4 แบบ คือ 1. กลุ่มใช้ในการทำการคำนวณ 2. กลุ่มใช้ในการเปรียบเทียบ 3. กลุ่มใช้ในการควบคุม และ 4. กลุ่มในการจัดการข้อมูล แสดงรายละเอียดในตารางที่ 2.2

Arithmetic:	add sub mul div mod
Logic:	and or xor eq ne lt le gt ge shl shr
Control:	jmp jt jf jal ret
Data:	ld st push pop

ตารางที่ 2.2 รายการ Instruction type

เมื่อใช้งานในระดับภาษา Assembly จะต้องแบ่งส่วนของรหัสข้อมูลเป็น 3 ส่วนหลัก ดังตารางที่ 2.3 แสดงการแบ่งส่วนรหัสข้อมูล โดยเริ่มจากส่วนของสัญลักษณ์ แล้วเป็นส่วนของรหัสคำสั่งที่ตำแหน่ง 100 และส่วนของข้อมูลที่ตำแหน่ง 5000

```
.symbol
...
.code 100
...
.data 5000
...
.end
```

ตารางที่ 2.3 การแบ่งส่วนรหัสข้อมูล

2.1.7 ภาษา Rz

ภาษา Rz พัฒนาเพื่อใช้ในการศึกษา โดยมีขนาดเล็กและลักษณะภาษาคล้ายกับภาษา C ที่ไม่มีการแบ่งชนิดของข้อมูลหรือมีข้อมูลเป็นได้เพียง Integer ชนิดเดียว

ตัวอย่างภาษา Rz แสดงในตารางที่ 2.4 แสดงตัวอย่างการเรียกการใช้งานแกนหน่วยประมวลผลทั้งสองแกน โดยการเริ่มการทำงานของแกนหลักเริ่มที่ฟังก์ชัน main() ให้เรียกคำสั่ง syscall(15,0) เพื่อทำการปลุกแกนหน่วยประมวลผลที่สองซึ่งเริ่มทำงานที่ฟังก์ชัน main2()

```
// Example
// Dual core demo
core1()
    i = 10
    while( i < 100 )
        print(i, “ “)
        i = i + 1

core2()
    i = 1010
    while( i < 1100 )
        print(i , “ “)
        i = i + 1

main2()
    core2()

main()
    syscall(15,0)
    core1()
```

ตารางที่ 2.4 ตัวอย่างภาษา Rz

2.2 งานวิจัยที่เกี่ยวข้อง

ในการศึกษางานวิจัยเรื่องการจัดการตารางงานบนหน่วยประมวลผลชนิดหลายแกน James Mistry [8] ได้เสนอการพัฒนาระบบปฏิบัติการ FreeRTOS ให้สนับสนุนหน่วยประมวลผลหลายแกน โดยพัฒนาบนหน่วยประมวลผล MicroBlaze (FPGAs) มีรายการความต้องการในการพัฒนาเช่น สามารถจัดตารางงานสำหรับหน่วยประมวลผลสองแกน มี API สำหรับการเรียกใช้ mutual exclusion เพื่อการจัดการข้อมูลที่ใช้ร่วมกันระหว่างส่วนงานและรหัสคำสั่งต้องง่ายต่อการทำความเข้าใจ สำหรับการออกแบบการจัดตารางงานได้พิจารณาจากความสัมพันธ์แกนประมวลผล (Core Affinity) และระดับความสำคัญของส่วนงาน มีการกำหนดเลขแกนหน่วยประมวลผลให้กับส่วนงาน เพื่อระบุว่าทำงานบนหน่วยประมวลผลใด ในการจัดตารางงานจะเลือกส่วนงานที่มีระดับความสำคัญสูงสุดและมีความสัมพันธ์กับแกนประมวลผลมาทำงานก่อน ไม่มีการย้ายการทำงานของส่วนงานจากแกนหน่วยประมวลผลที่ถูกเริ่มทำงาน ในการเข้าถึงข้อมูลที่ใช้งานร่วมกันจะใช้ Memory barriers

การทดลองได้ทำการทดสอบโดยสร้างส่วนงานมา 3 ส่วนงานให้ส่วนงานหนึ่งและส่วนงานสองใช้ mutex ทำการบวกเลข โดยข้อมูลใช้งานร่วมกันระหว่างส่วนงาน ให้ส่วนงานสามทำการตรวจสอบการทำงานและแสดงผลการทำงานหลังส่วนงานหนึ่งและส่วนงานสองจบการทำงาน

Nicolas Navet [9] ได้เสนอการจัดตารางงานที่ใช้งานในงานขนาดใหญ่ที่มีจำนวนส่วนโปรแกรมมากมาย การจัดตารางงานจะทำการเลือกแกนหน่วยประมวลผลโดยใช้ค่าการใช้ประโยชน์ของแกนหน่วยประมวลผล (CPU Utilization Rate) จะเพิ่มส่วนงานให้กับแกนหน่วยประมวลผลที่มีค่าการใช้ประโยชน์ของแกนหน่วยประมวลผลที่น้อยและพิจารณาจากกลุ่มของส่วนงานมีความสัมพันธ์กับแกนหน่วยประมวลผล โดยกลุ่มของส่วนงานจะผ่านการแบ่งแยกการพิจารณาความเกี่ยวข้องกับแกนหน่วยประมวลผลก่อน การทดลองได้พัฒนาบน AUTOSAR เป็นระบบปฏิบัติการที่นิยมใช้ในอุตสาหกรรมยานยนต์

Hiroyuki Tomiyama [10] ได้เสนอการพัฒนาระบบปฏิบัติการ μITRON ให้สามารถทำงานบน Altera NiosII/s (FPGAs) หน่วยประมวลผลแบบไม่สมมาตร โดยอธิบายรายการความต้องการของการพัฒนาระบบปฏิบัติการ ให้มีการศึกษาหน่วยประมวลผลเป้าหมายว่ามีลักษณะการทำงานอย่างไร มีการติดต่อระหว่างแกนหน่วยประมวลผลเช่นไร อธิบายหลักการและเทคนิคในการพัฒนาระบบปฏิบัติการมีการพัฒนา Lock Unit โดยใช้ Mutual exclusion เพื่อใช้งานข้อมูลที่ถูกใช้งานร่วมการระหว่างส่วนงาน การทดลองได้วัดขนาดรหัสคำสั่งที่เพิ่มขึ้นคือ 60% และทดลองประสิทธิภาพการทำงานจากเวลาการทำงาน โดยเปรียบเทียบการทำงานของระบบปฏิบัติการที่ยังไม่ได้ถูกพัฒนากับการทำงานของระบบปฏิบัติการที่ถูกพัฒนาแล้ว โดยดูจากระยะเวลาระหว่างการเรียกสัญญาณในการเรียกส่วนงาน

Diana Bautista [11] แสดงการเลือกแกนหน่วยประมวลผลจากการพิจารณาค่า λ คือค่าอัตราส่วนงานต่อเวลา หรือ $1/\lambda$ เป็นเวลาที่หนึ่งส่วนงานใช้บนแกนหน่วยประมวลผล โดยพิจารณาเลือกแกนหน่วยประมวนผลที่มีการถูกใช้งานน้อยกว่าเพื่อปรับเปลี่ยนระดับแรงดันไฟฟ้าและความถี่ของสัญญาณนาฬิกา วัตถุประสงค์เพื่อการประหยัดพลังงานที่ใช้บนหน่วยประมวลผล ทำการทดลองโดยจำลองเหตุการณ์การทำงานโดยมีภาระงานจากการคำนวณเช่น การคำนวณ CRC การบีบอัดไฟล์เสียงและการคลายไฟล์เสียง ได้แสดงผลการทดลองไว้ว่าสามารถประหยัดพลังงานได้ 34% ถึง 74%

บทที่ 3
วิธีการดำเนินการวิจัย

3.1 แนวความคิด

ตัวจัดการตารางงานต้องกระจายงานให้ทั่วถึงโดยการปรับดุลจำนวนส่วนงานระหว่างแกนหน่วยประมวลผล โดยมีข้อสมมุติฐานเบื้องต้นคือเมื่อให้งานกับแกนใดไปแล้วจะไม่มีการย้ายงานไปแกนอื่น ทั้งนี้เพื่อให้ระบบตอบสนองในเวลาจริงได้เช่นเดิม จำนวนงานทั้งหมดถูกกำหนดไว้ก่อนเริ่มงานแล้ว ตามข้อกำหนดเดิมของระบบปฏิบัติการ Micrium µC/OS-III

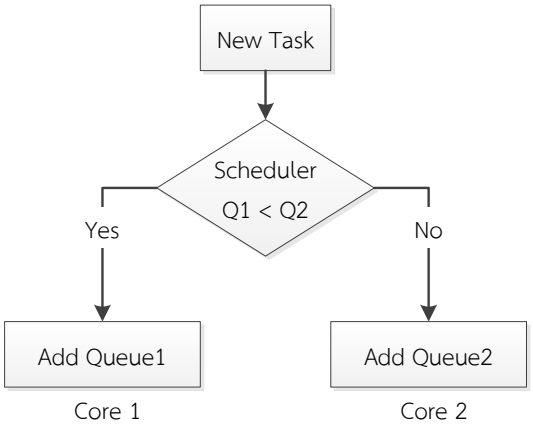
ตัวจัดการตารางงานจะทำงานบนแกนหน่วยประมวลผลหลัก มีหน้าที่จัดการตารางงานเพียงตัวเดียว ซึ่งลดความซับซ้อนของการทำงานและเพิ่มเสถียรภาพของระบบ

3.2 การออกแบบ

ออกแบบการจัดตารางงานให้ตัวจัดการตารางงานทำหน้าที่เลือกส่วนงานที่เหมาะสมทำงานบนแต่ละแกนหน่วยประมวลผล

3.2.1 วิธีการกระจายงาน

การกระจายงานต้องทราบภาระงานของทุกแกนก่อน ดังนั้นต้องเพิ่มรายการสถานะพร้อมทำงานตามจำนวนแกนที่เพิ่มขึ้นหรือแยกรายการสถานะพร้อมทำงานเป็นของแต่ละแกนหน่วยประมวลผล ตัววัดที่ใช้บอกภาระงานคือ จำนวนงานในรายการพร้อมทำงานของแกนนั้น ๆ โดยจะเลือกเพิ่มในแกนหน่วยประมวลผลที่มีจำนวนงานน้อยกว่าและเลือกเพิ่มในแกนหน่วยประมวลหลักก่อน ถ้าทุกแกนหน่วยประมวลผลมีจำนวนงานเท่ากัน โดยภาพที่ 3.1 แสดงตัวอย่างการกระจายงานของตัวจัดตารางงานที่ทำงานบนหน่วยประมวลผลสองแกน



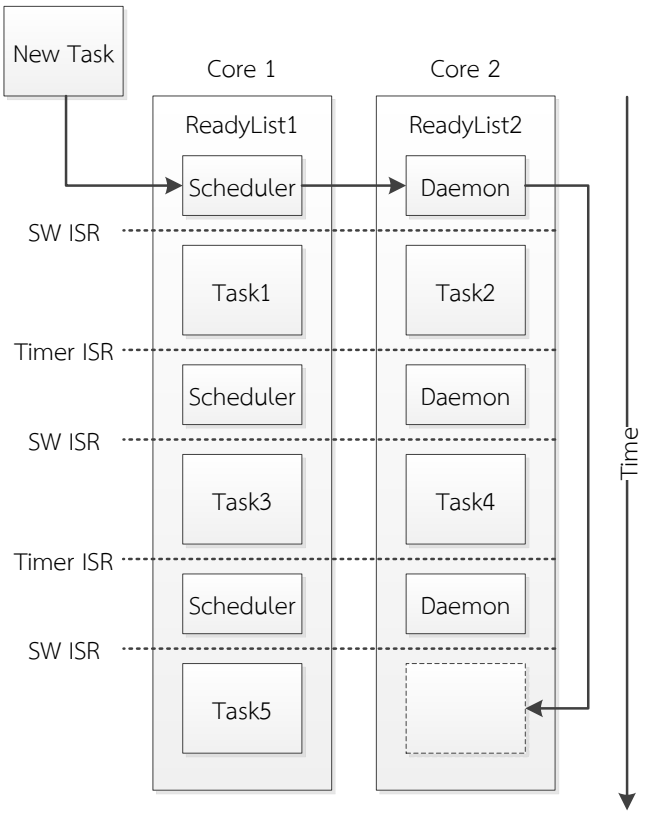
ภาพที่ 3.1 การกระจายงานของตัวจัดตารางงาน

3.2.2 วิธีการเพิ่มส่วนงาน

การเพิ่มส่วนงานจะทำหลังจากผ่านกระบวนการกระจายงาน ทำงานโดยตัวจัดตารางงานซึ่งทำงานบนหน่วยประมวลผลหลักเท่านั้น จึงสร้างตัวจัดตารางงานเป็นส่วนงานชื่อ Scheduler Task ให้ทำงานบนแกนประมวลผลหลัก ทำหน้าที่เพิ่มส่วนงานไปยังแต่ละรายการสถานะพร้อมทำงาน ทั้งเพิ่มส่วนงานบนหน่วยประมวลผลหลักและหน่วยประมวลผลอื่น ๆ

ในแกนหน่วยประมวลผลอื่น ๆ สร้างส่วนงานชื่อ Daemon Task ทำหน้าที่รับคำสั่งเพิ่มส่วนงานจากหน่วยประมวลผลหลัก ให้ทำงานเป็นระยะอยู่ตลอดเวลา เมื่อได้รับคำสั่งจะทำการเพิ่มส่วนงานในรายการสถานะพร้อมทำงานของแกนหน่วยประมวลผลที่ Daemon Task นั้นกำลังทำงานอยู่

เนื่องจากแต่ละแกนหน่วยประมวลผลมีรายการสถานะพร้อมทำงานของตนเอง แต่ละส่วนงานจะสลับทำงานตามรายการส่วนงานในรายการสถานะพร้อมทำงานโดย Scheduler Task และ Daemon Task จะสลับเป็นส่วนงานอื่นด้วยสัญญาณขัดจังหวะชนิดซอฟต์แวร์ ส่วนงานอื่นจะใช้สัญญาณขัดจังหวะชนิดนับเวลาเป็นตัวทำงานในส่วนการสลับส่วนงานดังภาพที่ 3.2



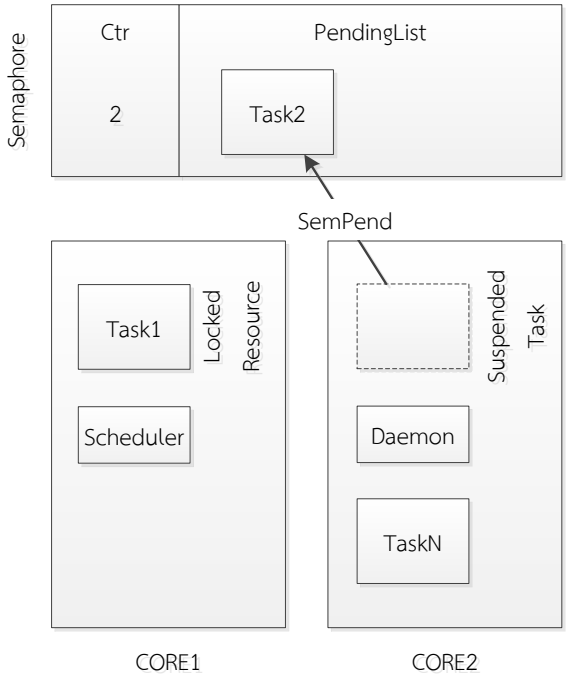
ภาพที่ 3.2 วิธีการเพิ่มส่วนงาน

3.2.3 วิธีจัดการทรัพยากร

การป้องกันการใช้ข้อมูลร่วมกันระหว่างแกน จะทำการพัฒนา Semaphore ให้สามารถใช้งานระหว่างแกนหน่วยประมวล

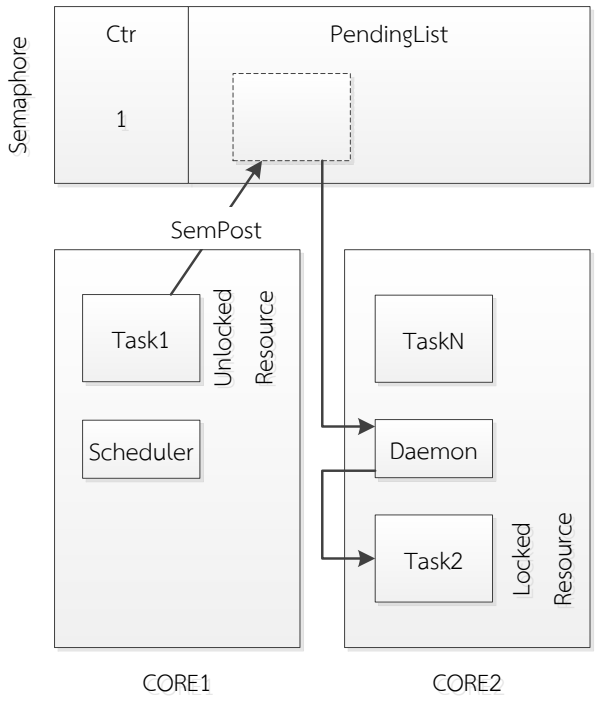
การพัฒนา Semaphore ให้สามารถทำงานบนหน่วยประมวลผลหลายแกน ดังภาพที่ 3.3 - 3.4 แสดงการใช้งาน Semaphore เพื่อใช้ข้อมูลร่วมกันระหว่างส่วนงานที่หนึ่งและส่วนงานที่สองซึ่งทั้งสองส่วนงานอยู่บนแกนหน่วยประมวลผลต่างกัน

จากภาพที่ 3.3 โดยให้ส่วนงานที่หนึ่งเข้าใช้งานข้อมูลก่อน เริ่มจากส่วนงานที่หนึ่งจะทำการป้องกันข้อมูลระหว่างที่กำลังใช้ข้อมูล ส่วนงานที่สองต้องการใช้ข้อมูลเหมือนกันก็จะต้องรอโดยถูกเปลี่ยนให้ตัวเองมีสถานะรอเรียกและย้ายไปใส่ในรายการรอเรียกใช้งานของ Semaphore



ภาพที่ 3.3 การใช้งาน Semaphore เมื่อเข้าสู่รายการรอเรียกใช้งาน

จากภาพที่ 3.4 เมื่อส่วนงานที่หนึ่งใช้ข้อมูลเสร็จสิ้นจะทำการปล่อยการป้องกันข้อมูลและทำการเปลี่ยนสถานะของส่วนงานในรายการรอเรียกใช้งานมาทำงานต่อซึ่งก็คือส่วนงานที่สองจะถูกเรียกมาใช้งาน โดยเปลี่ยนสถานะเป็นพร้อมทำงานและกลับไปอยู่รายการสถานะพร้อมทำงาน



ภาพที่ 3.4 การใช้งาน Semaphore เมื่อออกจากรายการรอเรียกใช้งาน

3.2.4 วิธีประสานระหว่างส่วนงาน

การประสานงานระหว่างส่วนงาน (Synchronization) และระหว่างแกนหน่วยประมวลผลจะใช้ Semaphore ในการจัดการให้แต่ละส่วนงานทำงานประสานกัน

3.2.5 วิธีติดต่อระหว่างส่วนงาน

ติดต่อระหว่างส่วนงานรวมทั้งระหว่างแกนหน่วยประมวลผลจะใช้การส่งข้อมูล (Message passing) โดยข้อมูลที่จะจัดส่งเป็นข้อมูลที่ใช้ร่วมกันระหว่างส่วนงาน ดังตารางที่ 3.1 แสดงการวิธีติดต่อระหว่างส่วนงาน ซึ่งใช้ Semaphore ในการจัดการข้อมูลที่ส่งระหว่างส่วนงาน

Task1()	Task2()
OSSemPend(pSemEmpty)	OSSemPend(pSemFull)
OSSemPend(pSemMutex)	OSSemPend(pSemMutex)
send message	receive message
OSSemPost(pSemMutex)	OSSemPost(pSemMutex)
OSSemPost(pSemFull)	OSSemPost(pSemEmpty)

ตารางที่ 3.1 วิธีติดต่อระหว่างส่วนงาน

3.3 การพัฒนา

การพัฒนาระบบปฏิบัติการ Micrium µC/OS-III ให้สามารถทำงานบนหน่วยประมวลผลหลายแกน ในงานวิจัยนี้ทำการทดลองบนหน่วยประมวลผล S2 โดยมีขั้นตอนดังนี้

3.3.1 กระชับรหัสข้อมูลของระบบปฏิบัติการ Micrium µC/OS-III

ระบบปฏิบัติการ Micrium µC/OS-III เป็นระบบที่มีการให้บริการฟังก์ชันการใช้งานหลายอย่าง ซึ่งบางอย่างไม่ได้ใช้ในการทดลอง ดังนั้นเพื่อลดความยุ่งยากและซับซ้อนในการดำเนินการจึงจำเป็นต้องกระชับรหัสข้อมูลให้เหลือเฉพาะส่วนที่ใช้งานเท่านั้น ดังตารางที่ 3.2 เปรียบเทียบไฟล์ระหว่างก่อนและหลังกระชับรหัสข้อมูล

ไฟล์ก่อนกระชับรหัสข้อมูล	ไฟล์หลังกระชับรหัสข้อมูล
os_core.c	os_core.c
os_cpu_a.asm	os_cpu_a.asm
os_cpu_c.c	os_cpu_c.c
os_dbg.c	os_prio.c
os_flag.c	os_sem.c
os_int.c	os_task.c
os_mem.c	
os_msg.c	
os_mutex.c	
os_pend_multi.c	
os_prio.c	
os_q.c	
os_sem.c	
os_stat.c	
os_task.c	
os_tick.c	
os_time.c	
os_tmr.c	
os_var.c	

ตารางที่ 3.2 เปรียบเทียบไฟล์ระหว่างก่อนและหลังกระชับรหัสข้อมูล

3.3.2 ย้ายระบบปฏิบัติการ Micrium µC/OS-III

ทำการย้ายระบบปฏิบัติการ Micrium µC/OS-III ให้สามารถทำงานบนหน่วยประมวลผล S2 มีขั้นตอนดังนี้

1) แปลงรหัสข้อมูลของระบบปฏิบัติการ Micrium μC/OS-III

ระบบปฏิบัติการ Micrium μC/OS-III มีรหัสข้อมูลในรูปภาษา C ในขณะที่หน่วยประมวลผล S2 ใช้ภาษา Rz ดังนั้นการย้ายระบบจะเริ่มจากทำการแปลงรหัสข้อมูลที่อยู่ในรูปภาษา C เป็นภาษา Rz ก่อน

ภาษา Rz	ภาษา C

def cont 1
g_var1
g_var2[10]
 	
func1 (var1, var2)
l_var = 0
if (g_var1 < var1)
l_var = var1 + var2
else
l_var = g_var1 + g_var2[0]
ret l_var
	#define cont 1
	Int g_var1;
	Int g_var2[10];
	
	Int func1 (int var1, int var2) {
	Int l_var = 0;
	If (g_var1 < var1) {
	l_var = var1 + var2;
	else {
	l_var = g_var1 + g_var2[0];
	}
	return l_var;
	}

ตารางที่ 3.3 เปรียบเทียบลักษณะภาษา Rz และภาษา C

ตามตารางที่ 3.3 ตารางเปรียบเทียบลักษณะภาษา Rz และภาษา C โดยแสดงการประกาศค่าคงที่ การประกาศตัวแปร Global variable และแสดงลักษณะการใช้ฟังก์ชัน

ภาษา Rz	ภาษา C

def HeadPtr 0
def TailPtr 1
def NbrEntries 2
 	
list1[3]
 	
func ()
list1[NbrEntries] = 0
	struct {
	int *HeadPtr;
	int *TailPtr;
	int NbrEntries;
	} list1;
	
	void func (void) {
	list1.NbrEntries = 0;
	}

ตารางที่ 3.4 การเปลี่ยน Structure เป็น Array of integer

นอกจากภาษา Rz มีลักษณะภาษาที่แตกต่างจากภาษา C แล้ว ภาษา Rz สามารถประกาศตัวแปรได้แค่ Integer และ Array of integer เท่านั้น ดังนั้นจึงต้องเปลี่ยนการเรียกใช้ Structure เป็น Array of integer แทน รวมถึงต้องระวังการใช้งานตัวแปร Pointer ด้วย ตามตารางที่ 3.4 ตารางแสดงการเปลี่ยน Structure เป็น Array of integer

รายการ	ภาษา Rz	ภาษา C
User defined data types	No	Yes
Data types	Only Int	Various
Declaration of constant	def	#define
Array dimensions	1	>1
Block delimited	No	Braces({ })
Data structure	No	Yes
Pointer variables	No	Yes
Increase and decrease (++, --)	No	Yes
For loop	No	Yes
While loop	Yes	Yes
Declaration of local variables	Auto	Needed

ตารางที่ 3.5 รายการเปรียบเทียบภาษา Rz และภาษา C

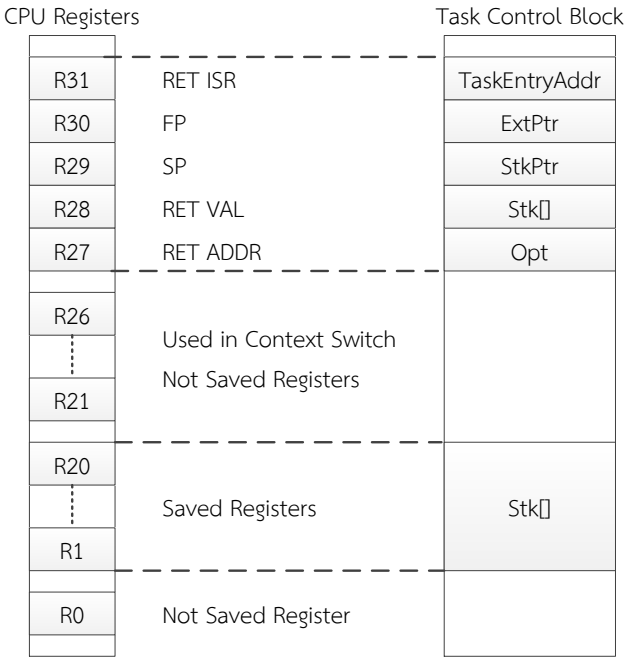
ภาษา Rz และภาษา C ยังมีความแตกต่างอื่น ๆ อีกเช่นการใช้เครื่องหมาย ++ และ -- ที่ภาษา Rz ไม่มี โดยดูรายการทั้งหมดได้ตามตางรางที่ 3.5 รายการเปรียบเทียบภาษา Rz และภาษา C

2) พัฒนาส่วนการเปลี่ยนส่วนงาน

การเปลี่ยนส่วนงานจะทำงานเมื่อเกิดสัญญาณขัดจังหวะเกิดขึ้นและต้องการเปลี่ยนส่วนงานที่กำลังทำงาน เมื่อได้รับสัญญาณขัดจังหวะจะบันทึกตำแหน่ง Stack Pointer ค่า CPU Registers และ Floating-Point Unit (FPU) Registers หรือสภาพการทำงานของหน่วยประมวลผลในขณะนั้นลง Task Control Block ของส่วนงานที่กำลังทำงาน จากนั้นจะเรียกข้อมูลจาก Task Control Block ของส่วนงานใหม่ขึ้นมาทำงานแทน

การสลับส่วนงานจำเป็นต้องรู้ลักษณะหน่วยประมวลผลที่ใช้งาน ในงานวิจัยนี้ใช้ตัวหน่วยประมวลผล S2 ซึ่งมี 32 Registers โดยตัวหน่วยประมวลผลใช้ R31 เก็บค่า Program Counter หรือตำแหน่งโปรแกรมที่กำลังทำงานก่อนถูกขัดจังหวะโดยสัญญาณขัดจังหวะ R30 เก็บค่า FP หรือตำแหน่งของ Activation Record สำหรับเก็บค่า Registers ของตัวแปรในฟังก์ชัน (Local Variables) R29 เก็บค่า SP หรือตำแหน่งของ Stack Pointer สำหรับเก็บค่า Registers ของ

พารามิเตอร์ของฟังก์ชัน R28 เก็บค่าตอบกลับจากฟังก์ชัน R27 เก็บค่าตำแหน่งโปรแกรมก่อนเรียกฟังก์ชัน ตัว Registers อื่น ๆ จะถูกใช้ในการทำงานทั่วไป



ภาพที่ 3.5 แสดง CPU Registers ที่บันทึกลง Task Control Block

ในส่วนนี้พัฒนาโดยใช้ภาษา Assembly เพราะสามารถจัดการ CPU Registers ได้ โดยเริ่มจากการบันทึกค่า CPU Registers ในการทำงานขณะนั้นลง Task Control Block ของส่วนงานที่กำลังทำงานอยู่ ทำการบันทึก R1 ถึง R20 และ R27 ถึง R31 ในส่วน R21 ถึง R26 จะไม่ถูกบันทึกเพราะสงวนไว้ใช้งานในระหว่างการเปลี่ยนส่วนงาน ดังภาพที่ 3.5 แสดง CPU Registers ที่บันทึกลง Task Control Block

หลังจากบันทึก CPU Registers ขณะที่ทำงานอยู่เสร็จแล้วก็จะทำการเรียกข้อมูลจาก Task Control Block ของส่วนงานใหม่ขึ้นมาทำงานแทน

```
; r21 = OSTCBCurPtr
ld r21 OSTCBCurPtr
; save sp
st sp @0 r21
; r21 = OSTCBHighRdyPtr
ld r21 OSTCBHighRdyPtr
; load sp
ld sp @0 r21
```

ตารางที่ 3.6 คำสั่งการบันทึกและการเรียกข้อมูลจาก Task Control Block

ตัวอย่างคำสั่งการบันทึกและการเรียกข้อมูลจาก Task Control Block เป็นภาษา Assembly แสดงในตารางที่ 3.6 คำสั่งการบันทึกและการเรียกข้อมูลจาก Task Control Block โดยเริ่มจากการเรียกค่าตำแหน่งข้อมูลจาก OSTCBCurPtr ตำแหน่งข้อมูลของ Task Control Block ที่กำลังทำงาน และทำการบันทึก SP ลงในช่องที่ 0 ณ ตำแหน่งของข้อมูล จากนั้นการเรียกค่าตำแหน่งข้อมูลจาก OSTCBHighRdyPtr ตำแหน่งข้อมูลของ Task Control Block ที่จะให้ทำงานแล้วทำการเรียก SP จากช่องที่ 0 ณ ตำแหน่งของข้อมูลใหม่ขึ้นมาแทน ดูรหัสการทำงาน ภาคผนวก ค ตารางที่ ค.6 หน้าที่ 57

3.3.3 พัฒนาการจัดตารางงาน

1) เพิ่มตัวแปรส่วนกลางตามจำนวนแกนหน่วยประมวลผล

แต่ละแกนจะมีหน่วยประมวลผลของตัวเอง จึงเพิ่มตัวแปรส่วนกลางได้แก่ รายการสถานะพร้อมทำงาน รายการลำดับความสำคัญและตัวชี้ Task Control Block โดยเพิ่มตามจำนวนแกนหน่วยประมวลผลที่เพิ่มขึ้น งานวิจัยนี้ทดลองบนหน่วยประมวลผล S2 เป็นหน่วยประมวลผลสองแกน ดังนั้นจึงเพิ่มแต่ละรายการมาหนึ่งรายการ ตามตารางที่ 3.7

แกนเดียว	สองแกน
OSRdyList	OSRdyList[2]
OSPrioTbl	OSPrioTbl[2]
OSTCBHighRdyPtr	OSTCBHighRdyPtr[2]
OSTCBCurPtr	OSTCBCurPtr[2]

ตารางที่ 3.7 การเพิ่มตัวแปรส่วนกลาง

2) พัฒนาส่วนเริ่มการทำงานของแกนหน่วยประมวลผลที่สอง

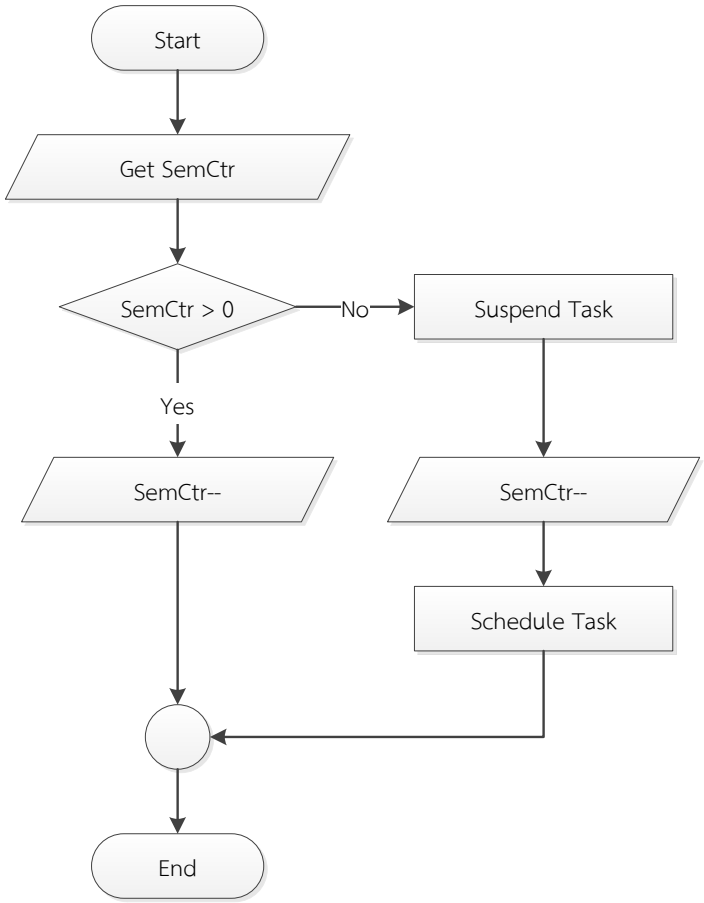
```
OSStartCore2()
    asm("mov r22 #1")
    asm("ld r21 @OSTCBCurPtr r22")
    asm("ld sp @0 r21")
    asm("ld fp @1 r21")
    asm("ld r22 @10 r21")
    asm("ret r22")

main2()
    OSStartCore2()
```

ตารางที่ 3.8 การเริ่มทำงานของแกนหน่วยประมวลผลแกนที่สอง

แกนที่สองของหน่วยประมวลผล S2 จะเริ่มทำงานโดยรับคำสั่งจากแกนหน่วยประมวลผลหลัก โดยจะเริ่มทำงานจากตำแหน่งหน่วยความจำที่ 30 หรือฟังก์ชัน main2() ในภาษา Rz แล้วจะเรียกส่วนงานในรายการสถานะพร้อมทำงานของแกนตัวเองขึ้นมาทำงาน ดังตารางที่ 3.8

3) พัฒนา Semaphore

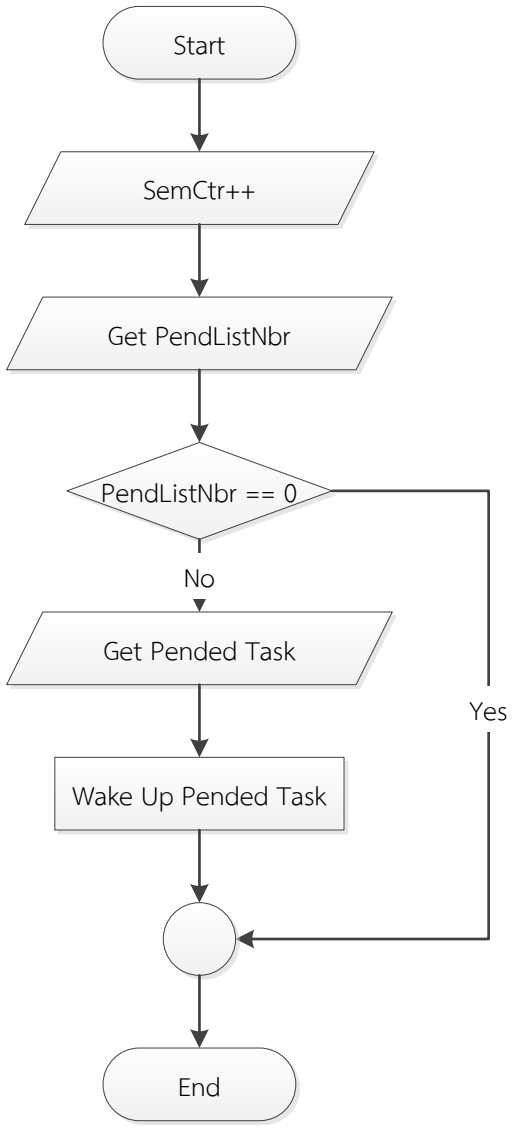
เริ่มจากการพัฒนาให้ Semaphore สามารถใช้งานบนหน่วยประมวลผลแกนเดียวก่อนคือให้สามารถใช้งานระหว่างสองส่วนงานบนแกนหน่วยประมวลผลเดียว โดยการพัฒนา Semaphore ทั่วไปจะใช้การที่เรียกว่า Exclusive access ซึ่งใช้ Atomic Instruction เฉพาะของหน่วยประมวลเป้าหมายเพื่อการเข้าถึงข้อมูลที่ใช้งานร่วมกันระหว่างแกนหน่วยประมวลผล แต่เนื่องจากหน่วยประมวลผล S2 ยังไม่มี Atomic Instruction ดังนั้นจึงใช้การเข้าสู่ Critical Section โดยการปิดสัญญาณขัดจังหวะและการหยุดการทำงานของอีกแกนในกรณีใช้ข้อมูลระหว่างแกน ใช้การเข้า Critical Section เพื่อทำการเข้าถึงตัวนับจำนวนและรายการรอการเรียกใช้งานของ Semaphore จากนั้นทำการพัฒนา Semaphore ให้สามารถใช้งานบนแกนหน่วยประมวลผลที่ต่างกันคือสองส่วนงานที่ทำงานอยู่บนคนละแกนหน่วยประมวลผล ดูรหัสการทำงาน ภาคผนวก ค ตารางที่ ค.4 หน้าที่ 50



ภาพที่ 3.6 ขั้นตอนการทำงานของ Semaphore ก่อนเข้าใช้ข้อมูล

ขั้นตอนการทำงานของ Semaphore ก่อนเข้าใช้ข้อมูลแสดงดังภาพที่ 3.6 โดยเริ่มจากการตรวจสอบเลขตัวนับจำนวนของ Semaphore ว่ามีค่ามากกว่าศูนย์หรือไม่ ถ้ามีค่ามากกว่าศูนย์แสดงว่าไม่มีการใช้งานอยู่ก่อนก็จะทำการลดเลขตัวนับจำนวนและจบการทำงาน แต่ถ้าค่ามีค่าน้อยกว่าหรือเท่ากับศูนย์แสดงว่ามีการใช้งานอยู่ก่อนก็จะทำการย้ายส่วนงานไปอยู่รายการเรียกใช้งานแล้วจะทำการลดเลขตัวนับจำนวนและทำการเรียกส่วนงานอื่นมาทำงานต่อไป จนกว่าจะมีการย้ายกลับและเรียกมาทำงานอีกครั้ง

ขั้นตอนการทำงานของ Semaphore ก่อนเข้าใช้ข้อมูลแสดงดังภาพที่ 3.7 โดยเริ่มจากการทำการเพิ่มตัวเลขจำนวนของ Semaphore แล้วตรวจสอบจำนวนส่วนงานที่อยู่ในรายการรอเรียกใช้งานว่ามีอยู่หรือไม่ ถ้าไม่มีก็จบการทำงาน แต่ถ้ามีส่วนงานเหลืออยู่ก็จะทำการเรียกส่วนงานในรายการขึ้นมาทำงาน

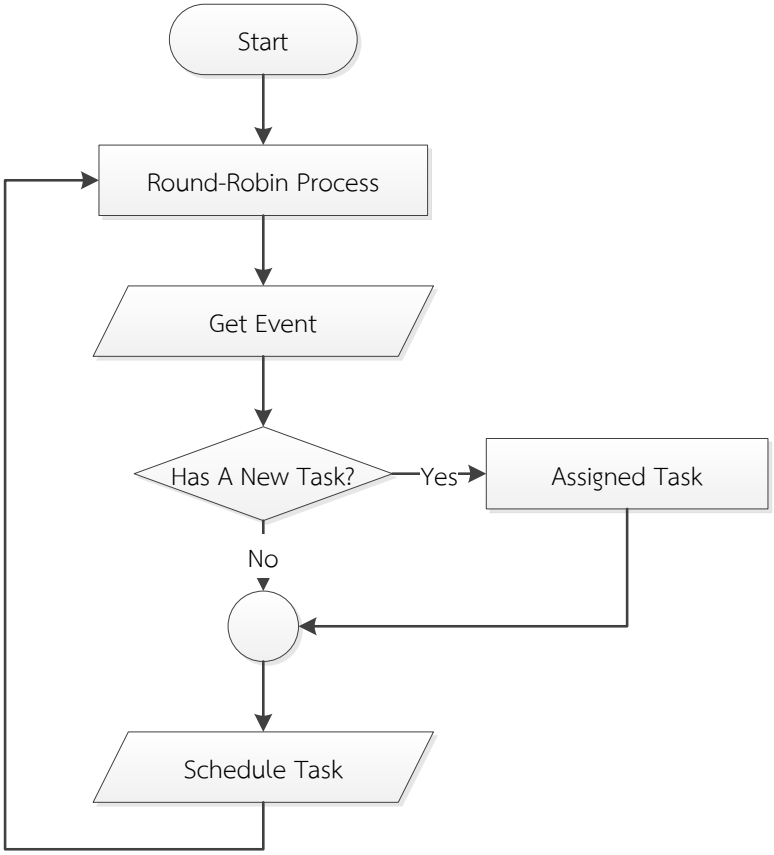


ภาพที่ 3.7 ขั้นตอนการทำงานของ Semaphore ออกจากใช้ข้อมูล

4) พัฒนา Scheduler Task

ส่วนงาน Scheduler Task เป็นส่วนงานในรายการสถานะพร้อมทำงานบนหน่วยประมวลผลแกนหลัก สร้างมาทดแทน Idle Task ของ Micrium µC/OS-III ที่มีอยู่เดิม โดยเพิ่มหน้าที่ให้จัดการตารางงานของระบบ กระจายส่วนงานหรือตัดสินใจเพิ่มส่วนงานให้ทำงานบนแกนหน่วยประมวลผลและสลับส่วนงานบนแกนหน่วยประมวลผลหลัก โดยพิจารณาเพิ่มส่วนงานจากจำนวนส่วนงานในรายการสถานะพร้อมทำงานของแกนที่มีจำนวนส่วนงานน้อยที่สุดเพื่อให้ทุกแกนมีภาระงานใกล้เคียงกัน ดูรหัสการทำงาน ภาคผนวก ค ตารางที่ ค.5 หน้าที่ 54

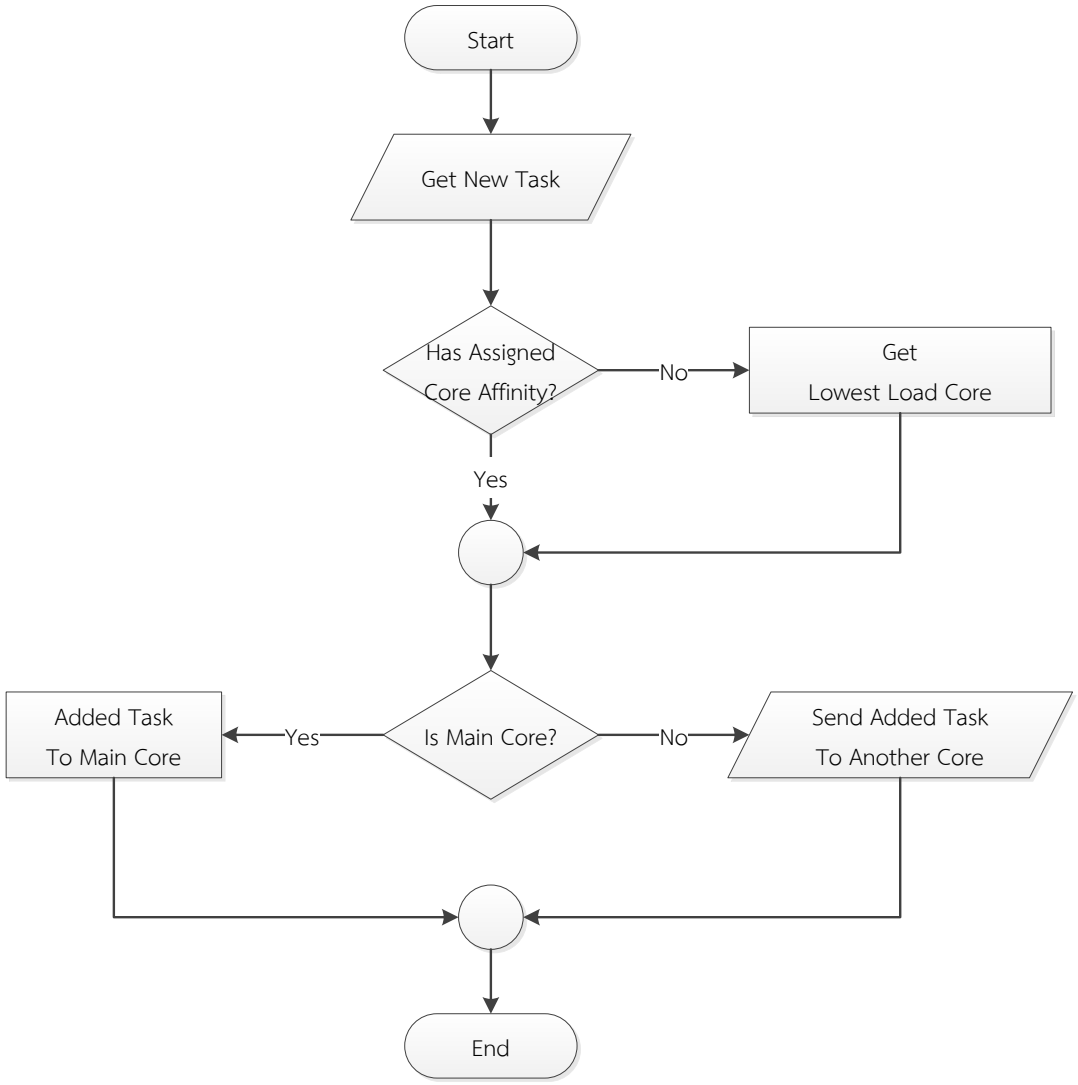
ขั้นตอนการทำงานของ Scheduler Task แสดงดังภาพที่ 3.8 โดยเริ่มจากกระบวนการทำ Round-Robin เพื่อสลับงานที่มีระดับความสำคัญเท่ากันมาแบ่งทำงาน จากนั้นทำการตรวจสอบว่ามีส่วนงานใหม่เกิดขึ้นหรือไม่ ถ้ามีส่วนงานใหม่จะเข้าสู่กระบวนการเพิ่มส่วนงานก่อนแล้วจึงเปลี่ยนส่วนงานเป็นส่วนงานถัดไป



ภาพที่ 3.8 ขั้นตอนการทำงานของ Scheduler Task

ขั้นตอนการเพิ่มส่วนงานของ Scheduler Task แสดงดังภาพที่ 3.9 เริ่มจากเช็คว่าส่วนงานได้ถูกเลือกแกนหน่วยประมวลผลไว้ก่อนหรือไม่ ถ้าไม่ได้ถูกเลือกไว้ก็จะทำการเลือกแกนจากจำนวนแกนที่มีภาระงานน้อยที่สุด จากนั้นจะทำการตรวจสอบว่าจะทำการเพิ่มส่วนงานในแกน

ใด ถ้าเป็นแกนหน่วยประมวลผลหลักก็ทำการเพิ่มในรายการสถานะพร้อมทำงานของหน่วยประมวลผลหลักแต่ถ้าเป็นแกนอื่นก็จะทำการส่งคำสั่งไปเพิ่มส่วนงานบนแกนหน่วยประมวลผลเป้าหมาย เพื่อรอให้ Daemon Task รับคำสั่งและทำการเพิ่มส่วนงานต่อไป



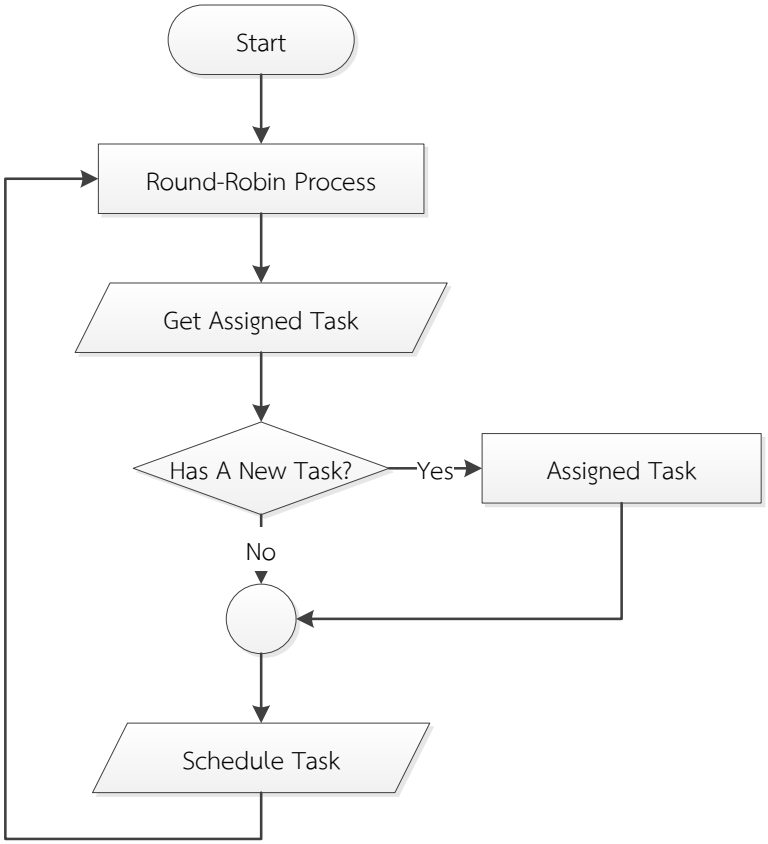
ภาพที่ 3.9 ขั้นตอนการเพิ่มส่วนงาน

การเพิ่มส่วนงานในแกนหน่วยประมวลผลอื่นจะใช้วิธีติดต่อระหว่างส่วนงานซึ่งใช้ Semaphore ในการป้องกันข้อมูลส่วนงานที่จะถูกเพิ่ม โดยเป็นการป้องกันการระหว่าง Scheduler Task และ Daemon Task ของแกนประมวลผลปลายทาง

5) พัฒนา Daemon Task

ส่วนงาน Daemon Task เป็นส่วนงานในรายการสถานะพร้อมทำงานบนหน่วยประมวลผลแกนอื่น ๆ สร้างมาทดแทน Idle Task ของ Micrium µC/OS-III ที่มีอยู่เดิมที่ไม่ใช่แกน

หน่วยประมวลผลหลัก มีหน้าที่เพิ่มส่วนงานในแกนหน่วยประมวลผลและสลับส่วนงานบนหน่วยประมวลผลที่ทำงานอยู่ ดูรหัสการทำงาน ภาคผนวก ค ตารางที่ ค.5 หน้าที่ 54



ภาพที่ 3.10 ขั้นตอนการทำงานของ Daemon Task

ขั้นตอนการทำงานของ Daemon Task แสดงดังภาพที่ 3.10 โดยเริ่มจากกระบวนการทำ Round-Robin เพื่อสลับงานที่มีระดับความสำคัญเท่ากันมาแบ่งทำงาน จากนั้นทำการตรวจสอบว่ามีสัญญาณส่วนงานใหม่เกิดขึ้นหรือไม่ ถ้ามีส่วนงานใหม่จะเข้าสู่กระบวนการเพิ่มส่วนงานก่อนแล้วจึงเปลี่ยนส่วนงานเป็นส่วนงานถัดไป สามารถดูรายละเอียดของรหัสข้อมูลเพิ่มเติมได้ที่ภาคผนวก ค

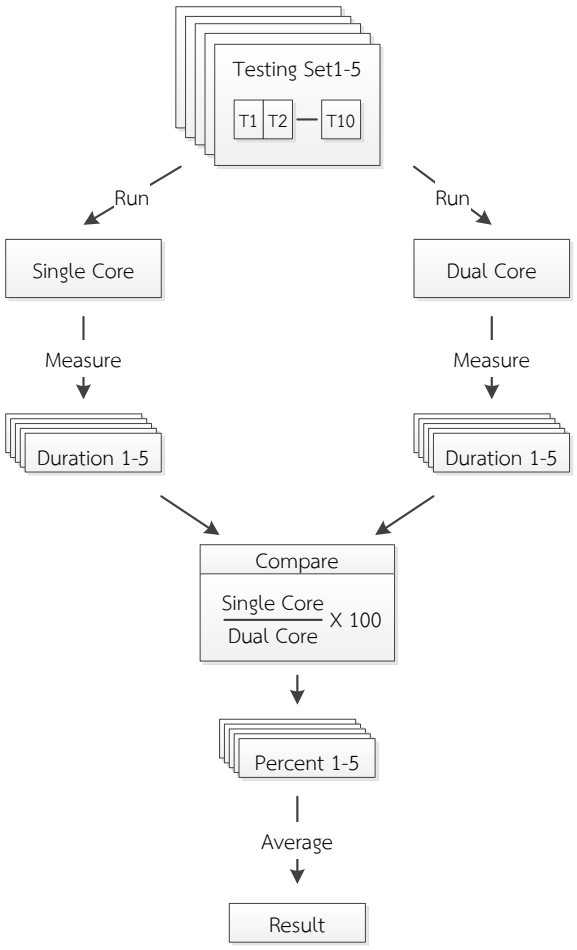
บทที่ 4
การทดลองและผลการทดลอง

4.1 เครื่องมือที่ใช้

การทดลองนี้ทำการทดลองบนโปรแกรมจำลองหน่วยประมวลผล S2 ซึ่งใช้เครื่องคอมพิวเตอร์ส่วนบุคคลที่มีหน่วยประมวลผลกลาง Interl Core i5-2410M CPU 2.3GHz หน่วยความจำ 4GB ระบบปฏิบัติการ Windows 7 Professional 64bits

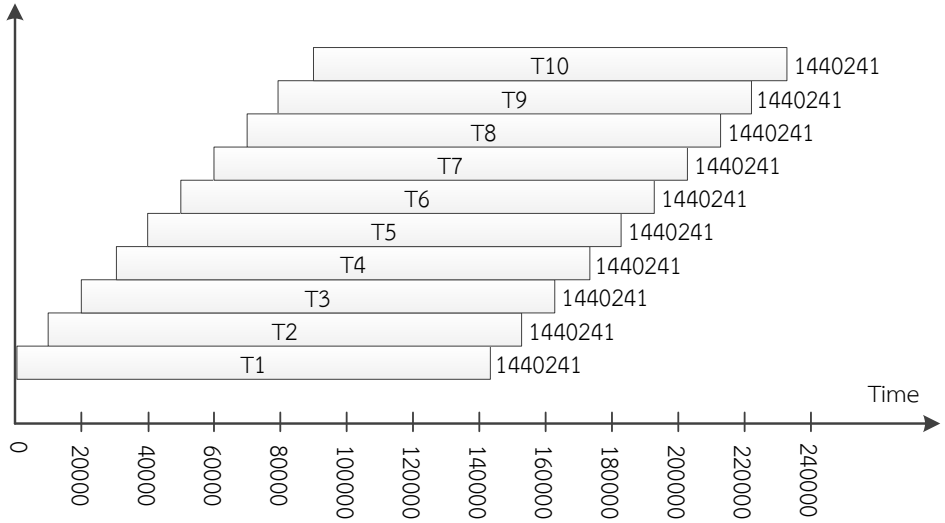
4.2 วิธีการทดลอง

ทดสอบการจัดตารางงานโดยเปรียบเวลาที่ใช้ในการทำงานบนหน่วยประมวลผลชนิดแกนเดียวและการทำงานบนหน่วยประมวลผลชนิดสองแกนจากชุดทดลอง 5 ชุด โดยแต่ละชุดมีส่วนงานจำนวน 10 ส่วนงาน ดังภาพที่ 4.1แสดงวิธีการทดลอง



ภาพที่ 4.1 วิธีการทดลอง

โดยแต่ละส่วนงานให้ทำการบวกนับเลขในจำนวนที่แตกต่างกัน เพื่อจำลองเวลาทำงานของแต่ละส่วนงานไม่เท่ากันและให้เริ่มการทำงานของแต่ละส่วนงานไม่พร้อมกัน เพื่อจำลองสถานการณ์การทำงานเป็นจำนวน 5 เหตุการณ์



ภาพที่ 4.2 ตัวอย่างชุดทดลองที่ 1

จากภาพที่ 4.2 ตัวอย่างชุดทดลองที่ 1 แสดงระยะเวลาทำงานและการเริ่มทำงานของแต่ละส่วนงาน จำนวนตัวเลขคือจำนวนสัญญาณนาฬิกาของระบบ แต่ละส่วนงานจะเกิดห่างกันที่ 10000 และด้านความยาวคือระยะเวลาการทำงานของแต่ละส่วนงาน สามารถดูรายละเอียดได้จากตารางที่ 4.1 รายละเอียดตัวอย่างชุดทดลองที่ 1 โดยชุดทดลองทั้งหมดแสดงไว้ในภาคผนวก ก

ส่วนงาน	เวลาเริ่มการทำงาน	ระยะเวลาทำงาน
1	0	1440241
2	10000	1440241
3	20000	1440241
4	30000	1440241
5	40000	1440241
6	50000	1440241
7	60000	1440241
8	70000	1440241
9	80000	1440241
10	90000	1440241

ตารางที่ 4.1 ตัวอย่างชุดทดลองที่ 1

4.3 ผลการทดลอง

ผลการทดสอบเป็นเลขจำนวนสัญญาณนาฬิกาที่ใช้ไปของแต่ละส่วนงาน โดยมีเวลาเริ่มส่วนงาน เวลาจบการทำงานและระยะเวลาทำงานของส่วนงานในชุดทดลอง แต่ละชุดทดลองจะทำการทดลองบนหน่วยประมวลผลแกนเดียวและหน่วยประมวลผลสองแกน

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	42759	18149395	18106636
2	53137	18187540	18134403
3	63447	18157688	18094241
4	73479	18217252	18143773
5	83484	18164148	18080664
6	93489	18193994	18100505
7	103494	18171145	18067651
8	113499	18209280	18095781
9	123504	18179478	18055974
10	133509	18200985	18067476

ตารางที่ 4.2 ผลการทดลองชุดทดลองที่ 1 บนหน่วยประมวลผลแกนเดียว

จากตารางที่ 4.2 ผลการทดลองชุดทดลองที่ 1 บนหน่วยประมวลผลแกนเดียว แสดงให้เห็นว่าส่วนงานที่ 1 เริ่มทำงานเป็นส่วนงานแรกที่ 42759 และส่วนงานที่ 4 จบการทำงานเป็นลำดับสุดท้ายที่ 18217252 ส่วนงานที่ 9 ใช้ระยะเวลาทำงานน้อยที่สุดที่ 18055974 และส่วนงานที่ 4 ใช้ระยะเวลาทำงานมากที่สุดที่ 18143773 โดยชุดการทดลองที่ 1 ทำงานบนหน่วยประมวลผลแกนเดียวได้ใช้ระยะเวลาทำงานไปทั้งหมด 18174493

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	43100	8971310	8928210
2	57939	10191058	10133119
3	63482	9009900	8946418
4	84598	10300700	10216102
5	94934	10365669	10270735
6	93830	9127485	9033655
7	103837	9088375	8984538
8	125393	10355005	10229612
9	135473	10343887	10208414
10	143876	9135529	8991653

ตารางที่ 4.3 ผลการทดลองชุดทดลองที่ 1 บนหน่วยประมวลผลสองแกน

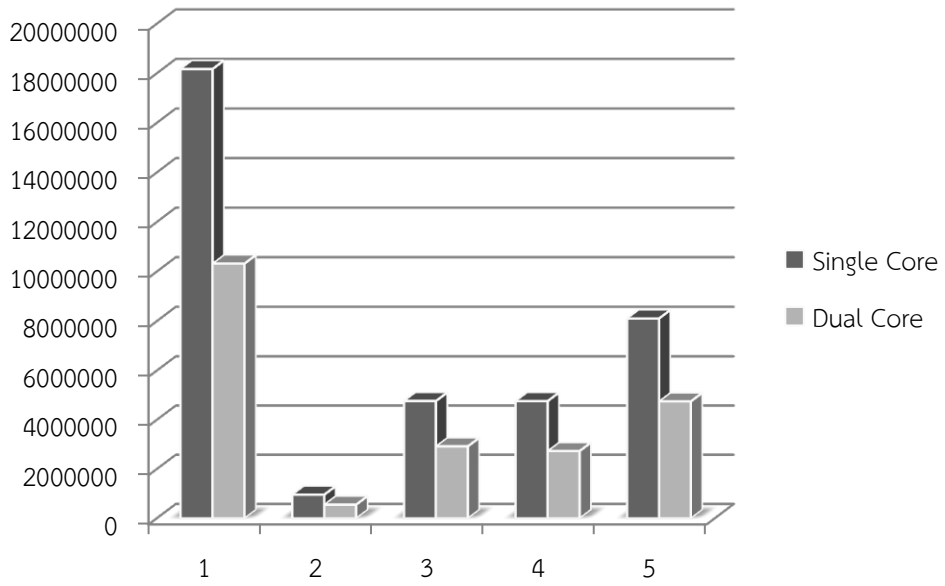
จากตารางที่ 4.3 ผลการทดลองชุดทดลองที่ 1 บนหน่วยประมวลผลสองแกน แสดงให้เห็นว่าส่วนงานที่ 1 เริ่มทำงานเป็นส่วนงานแรกที่ 43100 และส่วนงานที่ 5 จบการทำงานเป็นลำดับสุดท้ายที่ 10365669 ส่วนงานที่ 1 ใช้ระยะเวลาทำงานน้อยที่สุดที่ 8928210 และส่วนงานที่ 5 ใช้ระยะเวลาทำงานมากที่สุดที่ 10270735 โดยชุดการทดลองที่ 1 ทำงานบนหน่วยประมวลผลสองแกนได้ใช้ระยะเวลาทำงานไปทั้งหมด 10322569 ผลการทดลองทั้งหมดแสดงในภาคผนวก ข

ทำการเปรียบเทียบผลระยะเวลาการทำงานบนหน่วยประมวลผลแกนเดียวต่อการทำงานบนสองแกน ดังตารางที่ 4.4 แสดงให้ให้เห็นว่าชุดทดลองที่ 1 ให้ผลเปรียบเทียบระยะเวลาการทำงานที่มากที่สุดที่ 176.07 และชุดทดลองที่ 3 ให้ผลเปรียบเทียบระยะเวลาการทำงานที่น้อยที่สุดที่ 163.00

ลำดับ	แกนเดียว	สองแกน	เปรียบเทียบผล
1	18174493	10322569	176.07
2	959740	554791	172.99
3	4766907	2924542	163.00
4	4763027	2731129	174.40
5	8108597	4750004	170.71

ตารางที่ 4.4 เปรียบเทียบการทำงานบนหน่วยประมวลผลแกนเดียวและสองแกน

นำผลการทดลองที่ได้มาทำการเปรียบเทียบโดยกราฟ ดังภาพที่ 4.3 กราฟแท่งเปรียบเทียบเวลาในการทำงาน



ภาพที่ 4.3 กราฟแท่งเปรียบเทียบเวลาในการทำงาน

นำผลการเปรียบเทียบการทำงานของหน่วยประมวลผลแกนเดียวและสองแกนของทั้ง 5 ชุด มาเฉลี่ยกัน ค่าที่ได้เป็นประสิทธิภาพของการทำงานของหน่วยประมวลสองแกนเปรียบเทียบกับหน่วยประมวลผลแกนเดียวมีค่าเพิ่มขึ้นเป็น 171 เปอร์เซ็นต์

บทที่ 5
สรุปผลวิจัยและข้อเสนอแนะ

5.1 สรุปผลวิจัย

งานวิจัยนี้นำเสนอวิธีการจัดตารางงานที่สามารถทำงานบนหน่วยประมวลผลหลายแกน เพื่อให้ระบบปฏิบัติการแบบเวลาจริงใช้งานทรัพยากรของหน่วยประมวลผลที่มีจำนวนแกนที่เพิ่มขึ้นอย่างคุ้มค่า โดยมีแนวคิดให้มีการกระจายงานให้ทั่วถึง ใช้การจัดตารางงานแบบการปรับดุลจำนวนส่วนงานระหว่างแกนหน่วยประมวลผล เพื่อลดความซับซ้อนและเพิ่มเสถียรภาพให้กับระบบ ได้ออกแบบให้ตัวจัดตารางงานเป็นส่วนงานหนึ่งให้ชื่อว่า Schedule Task ทำงานบนแกนหน่วยประมวลผลหลักและให้ส่วนงานที่ชื่อ Daemon Task รับคำสั่งจากตัวจัดตารางงานและทำงานบนแกนหน่วยประมวลผลอื่นๆ

การพัฒนาได้นำต้นแบบจากระบบปฏิบัติการ Micrium µCOS-III มาทำการพัฒนาให้สามารถทำงานบนหน่วยประมวลผล S2 ซึ่งเป็นหน่วยประมวลผลสองแกน โดยทำการย้ายระบบปฏิบัติการ Micrium µCOS-III ที่มีรหัสข้อมูลอยู่ในรูปภาษา C ภาษา Assembly และยังไม่สนับสนุนการทำงานบนหน่วยประมวลผลหลายแกน ให้อยู่ในรูปภาษา Rz พัฒนาให้สามารถทำงานบนหน่วยประมวลผล S2 หรือหน่วยประมวลผลสองแกนและพัฒนาส่วนการจัดการทรัพยากรระหว่างแกนหน่วยประมวลผล โดยมีอุปสรรค์การพัฒนาที่เครื่องมือในการแปลโปรแกรมจากภาษา Rz เป็นภาษา Assembly ที่ยังขาดตัวโปรแกรมตรวจแก้จุดบกพร่องจึงต้องทำการพัฒนาอย่างระมัดระวังและแบ่งการพัฒนาเป็นทีละส่วน

การทดลองได้ทำการทดลองบนชุดทดลอง 5 ชุด โดยแต่ละชุดทดลองมีส่วนงาน 10 งาน ให้ส่วนงานมีเวลาเริ่มการทำงานและระยะเวลาการทำงานที่แตกต่างกัน ทำการวัดระยะเวลาการทำงานของแต่ละชุดทดลองที่ทำงานบนหน่วยประมวลผลแกนเดียวและหน่วยประมวลผลสองแกน แล้วทำการเปรียบเทียบเวลาการทำงานที่ได้

เมื่อนำผลการเปรียบเทียบเวลาการทำงานบนหน่วยประมวลผลแกนเดียวและหน่วยประมวลผลหลายแกนของชุดทดลองทั้งหมดมาเฉลี่ยกัน ผลคือระบบมีประสิทธิภาพเพิ่มขึ้น 171 เปอร์เซ็นต์ ซึ่งค่าที่ได้ยังห่างจากค่าในอุดมคติที่ 200 เปอร์เซ็นต์ เพราะยังมีการใช้เวลาในกระบวนการจัดตารางงานและการสลับเปลี่ยนส่วนงาน

5.2 ข้อเสนอแนะ

การพัฒนาต่อไปควรทดลองกับหน่วยประมวลผลที่มีจำนวนแกนมากขึ้น ลดเวลาในกระบวนการจัดตารางงานและพัฒนาส่วนสนับสนุนอื่น ๆ เช่นส่วนการจัดการเวลา การรอการทำงานแบบนับเวลาและส่วนการทำงานร่วมกับสัญญาณขัดจังหวะอื่น ๆ

รายการอ้างอิง

[1] D. Geer. Industry Trends: Chip Makers Turn to Multicore Processors. IEEE Computer Society (2005) : 11-13.

[2] Jean J. Labrosse. µC/OS-III TM The Real-Time Kernel User's Manual. Micrium Press, 2010.

[3] Prabhas Chongstitvatana. S2: A Hypothetical 32-bit Processor Version 3 [Online]. 2013. Avialable from : http://www.cp.eng.chula.ac.th/faculty/pjw/project/s2/s23.htm [2013,April,28]

[4] Robert I. Davis and Alan Burns. A Survey of Hard Real-Time Scheduling for Multiprocessor System. ACM Computing Surveys (2011)

[5] John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson and Sanjoy Baruah. A Categorization of Multiprocessor Scheduling Problems and Algorithms. In Handbook on Scheduling Algorithms, Methods, and Models. Boca : Chapman Hall/CRC, 2004.

[6] Atsushi Hasegawa. Renesas' Multi-Core Technology [Online]. 2013. Avialable from : http://www.renesas.com/products/mpumcu/multi_core/child/multicore.jsp [2013,April,28]

[7] Qing Li and Caroline Yao. Real-Time Concepts for Embedded Systems. CMP Books, 2003.

[8] James Mistry, Matthew Naylor and Jim Woodcock. Adapting FreeRTOS for Multicore: an Experience Report. John Wiley & Sons, Ltd : 2010.

[9] Nicolas Navet, Aurélien Monot, Bernard Bavouxy and Françoise Simonot-Lion. Multi-Source and Multicore Automotive ECUs - OS Protection Mechanisms and Scheduling. In ISIE 2010, pp. 3734-3741. 4-7 July 2010. Italy, 2010.

[10] Hiroyuki Tomiyama, Shinya Honda and Hiroaki Takada. Real-Time Operating Systems for Multicore Embedded Systems. In ISOCC 2008, pp. I-62 – I-67. 24-25 November 2008. Korea, 2008

[11] Diana Bautista, Julio Sahuquillo, Houcine Hassan, Salvador Petit and Jos´e Duato. A Simple Power-Aware Scheduling for Multicore Systems when Running Real-Time Applications. In IPDPS 2008, pp. 1-7. 14-18 April 2008. USA, 2008

ภาคผนวก

ภาคผนวก ก

ชุดส่วนงานสำหรับทดลอง

รายละเอียดของชุดทดลองที่ไว้สำหรับทำการทดลองแบ่งเป็น 5 ตารางโดยมีรายละเอียดเกี่ยวกับเวลาเริ่มการทำงานและระยะเวลาทำงานของส่วนงานทั้งหมดในชุดทำงาน

ตารางที่ ก.1 ตัวอย่างชุดทดลองที่ 1

ส่วนงาน	เวลาเริ่มการทำงาน	ระยะเวลาทำงาน
1	0	1440241
2	10000	1440241
3	20000	1440241
4	30000	1440241
5	40000	1440241
6	50000	1440241
7	60000	1440241
8	70000	1440241
9	80000	1440241
10	90000	1440241

ตารางที่ ก.2 ตัวอย่างชุดทดลองที่ 2

ส่วนงาน	เวลาเริ่มการทำงาน	ระยะเวลาทำงาน
1	0	72241
2	10000	72241
3	20000	72241
4	30000	72241
5	40000	72241
6	50000	72241
7	60000	72241
8	70000	72241
9	80000	72241
10	90000	72241

ตารางที่ ก.3 ตัวอย่างชุดทดลองที่ 3

ส่วนงาน	เวลาเริ่มการทำงาน	ระยะเวลาทำงาน
1	0	72241
2	30000	72241
3	60000	216241
4	90000	216241
5	120000	360241
6	150000	360241
7	180000	504241
8	210000	504241
9	240000	720241
10	270000	720241

ตารางที่ ก.4 ตัวอย่างชุดทดลองที่ 4

ส่วนงาน	เวลาเริ่มการทำงาน	ระยะเวลาทำงาน
1	0	720241
2	30000	720241
3	60000	504241
4	90000	504241
5	120000	360241
6	150000	360241
7	180000	216241
8	210000	216241
9	240000	72241
10	270000	72241

ตารางที่ ก.5 ตัวอย่างชุดทดลองที่ 5

ส่วนงาน	เวลาเริ่มการทำงาน	ระยะเวลาทำงาน
1	0	72241
2	30000	1224241
3	50000	648241
4	60000	432241
5	90000	216241
6	110000	792241
7	120000	144241
8	160000	1440241
9	190000	864241
10	200000	576241

ภาคผนวก ข

ผลการทดลองของชุดส่วนงาน

รายละเอียดของผลการทดลองของชุดส่วนงานแบ่งเป็น 10 ตารางโดยมีรายละเอียดเกี่ยวกับเวลาเริ่มการทำงาน เวลาจบการทำงานและระยะเวลาการทำงานของส่วนงานทั้งหมดในชุดทำงาน

ตารางที่ ข.1 ผลการทดลองชุดทดลองที่ 1 บนหน่วยประมวลผลแกนเดียว

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	42759	18149395	18106636
2	53137	18187540	18134403
3	63447	18157688	18094241
4	73479	18217252	18143773
5	83484	18164148	18080664
6	93489	18193994	18100505
7	103494	18171145	18067651
8	113499	18209280	18095781
9	123504	18179478	18055974
10	133509	18200985	18067476

ตารางที่ ข.2 ผลการทดลองชุดทดลองที่ 1 บนหน่วยประมวลผลสองแกน

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	43100	8971310	8928210
2	57939	10191058	10133119
3	63482	9009900	8946418
4	84598	10300700	10216102
5	94934	10365669	10270735
6	93830	9127485	9033655
7	103837	9088375	8984538
8	125393	10355005	10229612
9	135473	10343887	10208414
10	143876	9135529	8991653

ตารางที่ ข.3 ผลการทดลองชุดทดลองที่ 2 บนหน่วยประมวลผลแกนเดียว

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	42759	943806	901047
2	53137	974681	921544
3	63447	949419	885972
4	73479	1002499	929020
5	83484	955601	872117
6	93489	982199	888710
7	103494	963119	859625
8	113499	995330	881831
9	123504	968764	845260
10	133509	987844	854335

ตารางที่ ข.4 ผลการทดลองชุดทดลองที่ 2 บนหน่วยประมวลผลสองแกน

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	43100	368194	325094
2	57939	403657	345718
3	63482	396756	333274
4	84598	521961	437363
5	94934	585048	490114
6	93830	523221	429391
7	103837	485780	381943
8	125393	597891	472498
9	135473	552978	417505
10	143876	528559	384683

ตารางที่ ข.5 ผลการทดลองชุดทดลองที่ 3 บนหน่วยประมวลผลแกนเดียว

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	42759	443962	401203
2	73138	704450	631312
3	103485	2245873	2142388
4	133525	2273455	2139930
5	163533	3384238	3220705
6	193538	3493654	3300116
7	223551	4256724	4033173
8	253559	4224169	3970610
9	283564	4809666	4526102
10	313572	4784630	4471058

ตารางที่ ข.6 ผลการทดลองชุดทดลองที่ 3 บนหน่วยประมวลผลสองแกน

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	43100	165039	121939
2	77696	209799	132103
3	103482	1088942	985460
4	145878	1204427	1058549
5	169594	1718192	1548598
6	203512	1727893	1524381
7	247471	2580978	2333507
8	287952	2653009	2365057
9	313900	2237970	1924070
10	359268	2967642	2608374

ตารางที่ ข.7 ผลการทดลองชุดทดลองที่ 4 บนหน่วยประมวลผลแกนเดียว

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	42759	4779847	4737088
2	73138	4805786	4732648
3	103485	4338668	4235183
4	133525	4329566	4196041
5	163533	3664151	3500618
6	193538	3610276	3416738
7	223551	2595870	2372319
8	253559	2604341	2350782
9	283564	1264612	981048
10	313572	1272159	958587

ตารางที่ ข.8 ผลการทดลองชุดทดลองที่ 4 บนหน่วยประมวลผลสองแกน

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	43100	2428629	2385529
2	77696	2774229	2696533
3	103482	2238273	2134791
4	145878	2570816	2424938
5	173838	1887262	1713424
6	217174	2174494	1957320
7	243882	1506056	1262174
8	288163	1671478	1383315
9	313903	804828	490925
10	359106	867638	508532

ตารางที่ ข.9 ผลการทดลองชุดทดลองที่ 5 บนหน่วยประมวลผลแกนเดียว

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	42759	514071	471312
2	53137	7805242	7752105
3	83478	5985882	5902404
4	113518	4317439	4203921
5	133165	2525483	2392318
6	163536	6910609	6747073
7	193541	1805021	1611480
8	213182	8151356	7938174
9	223518	6987577	6764059
10	253555	5488694	5235139

ตารางที่ ข.10 ผลการทดลองชุดทดลองที่ 5 บนหน่วยประมวลผลสองแกน

ส่วนงาน	เวลาเริ่มการทำงาน	เวลาจบการทำงาน	ระยะเวลาทำงาน
1	43100	208272	165172
2	57939	4411000	4353061
3	83483	3355679	3272196
4	115037	2245749	2130712
5	125377	1315931	1190554
6	153833	3899750	3745917
7	183874	1055610	871736
8	216710	4793104	4576394
9	292294	3974156	3681862
10	243524	3268161	3024637

ภาคผนวก ค

ตัวอย่างชุดรหัสข้อมูล

รายละเอียดของชุดรหัสข้อมูลที่ใช้ในงานวิจัย อยู่ในรูปภาษา Rz และรูปภาษา Assembly

ตารางที่ ค.1 ตัวอย่างชุดรหัสข้อมูลส่วนการจัดการระดับความสำคัญ

```
OS_PrioInsert(p_tcb, p_prio_tbl)
    prio          = 0
    bit           = 0
    prio_tmp      = 0
    prio          = p_tcb[OS_TCB_Prio]
    bit           = 1 << prio
    prio_tmp      = p_prio_tbl[0]
    asm("or r4 r4 r5")
    p_prio_tbl[0] = bit

OS_PrioRemove(p_tcb, p_prio_tbl)
    prio          = 0
    bit           = 0
    prio_tmp      = 0
    prio          = p_tcb[OS_TCB_Prio]
    bit           = 1 << prio
    prio_tmp      = p_prio_tbl[0]
    asm("xor r4 r4 #65535")
    asm("and r4 r4 r5")
    p_prio_tbl[0] = bit

CPU_CntLeadZeros(p_tbl)
    prio_tbl      = p_tbl
    i             = 0
    prio_tbl      = p_tbl >> i
    prio_tbl_tmp  = 0
    asm("and r4 r2 #1")
    while (prio_tbl_tmp == 0)
        i = i + 1
        if (i >= DEF_INT_CPU_NBR_BITS)
            return 0
        prio_tbl = p_tbl >> i
        asm("and r4 r2 #1")
    return i

OS_PrioGetHighest(p_tcb, p_tbl)
    prio = CPU_CntLeadZeros(*p_tbl
    return prio
```

ตารางที่ ค.2 ตัวอย่างชุดรหัสข้อมูลส่วนการจัดการรายการสถานะรอเรียกทำงาน

```
OS_RdyListMoveHeadToTail(p_rdy_list)
    if (p_rdy_list[OS_Rdy_NbrEntries] == 2 ) // SWAP the TCBs
        p_tcb1                      = p_rdy_list[OS_Rdy_HeadPtr]
        p_tcb2                      = p_rdy_list[OS_Rdy_TailPtr]
        p_tcb1[OS_TCB_PrevPtr]      = p_tcb2
        p_tcb1[OS_TCB_NextPtr]      = 0
        p_tcb2[OS_TCB_PrevPtr]      = 0
```

```
        p_tcb2[OS_TCB_NextPtr]      = p_tcb1
        p_rdy_list[OS_Rdy_HeadPtr]  = p_tcb2
        p_rdy_list[OS_Rdy_TailPtr]  = p_tcb1
    else
        // Move only if there are more than 2 OS_TCBs in the list
        if (p_rdy_list[OS_Rdy_NbrEntries] > 2 )
            p_tcb1                      = p_rdy_list[OS_Rdy_HeadPtr]
            p_tcb2                      = p_rdy_list[OS_Rdy_TailPtr]
            p_tcb3                      = p_tcb1[OS_TCB_NextPtr]
            p_tcb3[OS_TCB_PrevPtr]      = 0
            p_tcb1[OS_TCB_NextPtr]      = 0
            p_tcb1[OS_TCB_PrevPtr]      = p_tcb2
            p_tcb2[OS_TCB_NextPtr]      = p_tcb1
            p_rdy_list[OS_Rdy_HeadPtr]  = p_tcb3
            p_rdy_list[OS_Rdy_TailPtr]  = p_tcb1

OS_RdyListInsert(p_tcb, p_rdy_list, os_prio_cur)
    if (p_tcb[OS_TCB_Prio] == os_prio_cur)
        if ( p_rdy_list[OS_Rdy_NbrEntries] == 0 )
            p_rdy_list[OS_Rdy_NbrEntries]     = 1
            p_tcb[OS_TCB_NextPtr]             = 0
            p_tcb[OS_TCB_PrevPtr]             = 0
            p_rdy_list[OS_Rdy_HeadPtr]        = &p_tcb[0]
            p_rdy_list[OS_Rdy_TailPtr]        = &p_tcb[0]
        else
            p_rdy_list[OS_Rdy_NbrEntries]= p_rdy_list[OS_Rdy_NbrEntries]+1
            p_tcb[OS_TCB_NextPtr]             = 0
            p_tcb2                            = p_rdy_list[OS_Rdy_TailPtr]
            p_tcb[OS_TCB_PrevPtr]             = &p_tcb2[0]
            p_tcb2[OS_TCB_NextPtr]            = &p_tcb[0]
            p_rdy_list[OS_Rdy_TailPtr]        = &p_tcb[0]
    else
        if (p_rdy_list[OS_Rdy_NbrEntries] == 0)
            p_rdy_list[OS_Rdy_NbrEntries]     =  1
            p_tcb[OS_TCB_NextPtr]             = 0
            p_tcb[OS_TCB_PrevPtr]             = 0
            p_rdy_list[OS_Rdy_HeadPtr]        = &p_tcb[0]
            p_rdy_list[OS_Rdy_TailPtr]        = &p_tcb[0]
        else
            p_rdy_list[OS_Rdy_NbrEntries] = p_rdy_list[OS_Rdy_NbrEntries]+1
            p_tcb[OS_TCB_NextPtr]             = p_rdy_list[OS_Rdy_HeadPtr]
            p_tcb[OS_TCB_PrevPtr]             = 0
            p_tcb2                            = p_rdy_list[OS_Rdy_HeadPtr]
            p_tcb2[OS_TCB_PrevPtr]            = &p_tcb[0]
            p_rdy_list[OS_Rdy_HeadPtr]        = &p_tcb[0]

OS_RdyListRemove(p_tcb, p_rdy_list)
    p_tcb1      = p_tcb[OS_TCB_PrevPtr]
    p_tcb2      = p_tcb[OS_TCB_NextPtr]
    if (p_tcb1 == 0)
        if (p_tcb2 == 0)
            p_rdy_list[OS_Rdy_NbrEntries] = 0
            p_rdy_list[OS_Rdy_HeadPtr]    = 0
            p_rdy_list[OS_Rdy_TailPtr]    = 0
            core = p_tcb[OS_TCB_CoreRunning]
            p_prio_tbl  = OSPrioTbl[core]
            OS_PrioRemove(p_tcb, p_prio_tbl)
        else
            p_rdy_list[OS_Rdy_NbrEntries] = p_rdy_list[OS_Rdy_NbrEntries]-1
            p_tcb2[OS_TCB_PrevPtr]        = 0
            p_rdy_list[OS_Rdy_HeadPtr]    = p_tcb2
    else
        p_rdy_list[OS_Rdy_NbrEntries]     = p_rdy_list[OS_Rdy_NbrEntries]-1
        p_tcb1[OS_TCB_NextPtr]            = p_tcb2
```

```
        if (p_tcb2 == 0)
            p_rdy_list[OS_Rdy_TailPtr]    = p_tcb1
        else
            p_tcb2[OS_TCB_PrevPtr]        = p_tcb1
    p_tcb[OS_TCB_PrevPtr] = 0
    p_tcb[OS_TCB_NextPtr] = 0
```

ตารางที่ ค.3 ตัวอย่างชุดรหัสข้อมูลส่วนการจัดการส่วนงาน

```
OS_TaskInitTCB(p_tcb)
    p_tcb[OS_TCB_StkLimitPtr]           = 0
    p_tcb[OS_TCB_NextPtr]               = 0
    p_tcb[OS_TCB_PrevPtr]               = 0
    p_tcb[OS_TCB_TickNextPtr]           = 0
    p_tcb[OS_TCB_TickPrevPtr]           = 0
    p_tcb[OS_TCB_TickSpokePtr]          = 0
    p_tcb[OS_TCB_TaskEntryArg]          = 0
    p_tcb[OS_TCB_PendingDataTblPtr]     = 0
    p_tcb[OS_TCB_PendingOn]             = OS_TASK_PEND_ON_NOTHING
    p_tcb[OS_TCB_PendingStatus]         = OS_STATUS_PEND_OK
    p_tcb[OS_TCB_TaskState]             = OS_TASK_STATE_RDY
    p_tcb[OS_TCB_Prio]                  = OS_PRIO_INIT
    p_tcb[OS_TCB_Opt]                   = 0
    p_tcb[OS_TCB_PendingDataTblEntries]    = 0
    p_tcb[OS_TCB_TS]                    = 0
    p_tcb[OS_TCB_SemCtr]                = 0
    p_tcb[OS_TCB_TickCtrPrev]           = OS_TICK_TH_INIT
    p_tcb[OS_TCB_TickCtrMatch]          = 0
    p_tcb[OS_TCB_TimeQuantaCtr]         = 0
    p_tcb[OS_TCB_CoreAffinity]          = NO_CORE
    p_tcb[OS_TCB_CoreRunning]           = NO_CORE

OSTaskDel(p_tcb)
    core = p_tcb[OS_TCB_CoreRunning]
    if ( core == CORE1 )
        asm("trap r21 #20 ; CORE1_INT_DIS")
    else
        asm("trap r21 #22 ; CORE2_INT_DIS")
    OSTaskQty[core] = OSTaskQty[core] - 1
    p_rdy       = OSRdyList[core]
    p_rdy_list  = p_rdy[p_tcb[OS_TCB_Prio]]
    if ( p_tcb[OS_TCB_TaskState] != OS_TASK_STATE_RDY )
        if ( core == CORE1 )
            asm("trap r21 #21 ; CORE1_INT_EN")
        else
            asm("trap r21 #23 ; CORE2_INT_EN")
    else
        OS_RdyListRemove(p_tcb, p_rdy_list)
        OS_TaskInitTCB(p_tcb)
        p_tcb[OS_TCB_TaskState] = OS_TASK_STATE_DEL
        if ( core == CORE1 )
            OSTCBHighRdyPtr[CORE1] = OSSchedTaskTCBPtr
            OSPrioHighRdy[CORE1] = OSSchedTaskPrio
            asm("trap r21 #21 ; CORE1_INT_EN ")
            asm("trap r21 #16 ; SW_INT_CORE1")
        else
            OSTCBHighRdyPtr[CORE2] = OSDaemonTaskTCBPtr
            OSPrioHighRdy[CORE2] = OSDaemonTaskPrio
            asm("trap r21 #23")  // CORE2_INT_EN
            asm("trap r21 #17 ; SW_INT_CORE2")
```

```
OSTaskCreate(p_tcb, p_task, core, prio, p_stk_base, stk_size, p_ext)
    OS_TaskInitTCB(p_tcb)
    p_stk = &p_stk_base[0]
    while ( i < stk_size )              // 0 - 31 ( R0 - R31 )
        *p_stk  = 0
        p_stk   = p_stk + 1
        i = i + 1
    p_stk = &p_stk_base[0]
    p_stk = p_stk + 48                  // shift to 48
    p_sp = p_stk
    p_tcb[OS_TCB_TaskEntryAddr]     = p_task
    p_tcb[OS_TCB_TaskEntryArg]      = 0
    p_tcb[OS_TCB_NamePtr]           = 0
    p_tcb[OS_TCB_Prio]              = prio
    p_tcb[OS_TCB_StkPtr]            = p_sp
    p_tcb[OS_TCB_StkLimitPtr]       = 0
    p_tcb[OS_TCB_TimeQuanta]        = 0
    p_tcb[OS_TCB_ExtPtr]            = p_ext
    p_tcb[OS_TCB_StkBasePtr]        = p_stk_base
    p_tcb[OS_TCB_StkSize]           = stk_size
    p_tcb[OS_TCB_Opt]               = 0
    p_tcb[OS_TCB_CoreAffinity]      = core

Task1Start()
    T1Data          = 0
    core            = OS_TCB_Task1TCB[OS_TCB_CoreRunning]
    StartT1Time     = OS_TS_GET1(core)
    while ( T1Data <= T1DataMax )
        print("a:", T1Data)
        T1Data      = T1Data + 1
    OSTask1TCBPtr   = &OS_TCB_Task1TCB[0]
    EndT1Time       = OS_TS_GET1(core)
    OSTaskDel(OSTask1TCBPtr)

CreateTask()
    OSSchedTaskTCBPtr           = &OS_TCB_SchedTaskTCB[0]
    OSSchedTaskPtr              = &OS_SchedTask
    OSSchedTaskPrio             = OS_CFG_PRIO_MAX - 1
    OSSchedTaskCore             = CORE1
    OSCfg_SchedTaskStkBasePtr   = &OSCfg_SchedTaskStk[0]
    OSCfg_SchedTaskFpBasePtr    = &OSCfg_SchedTaskFp[0]
    OSTaskCreate(OSSchedTaskTCBPtr, OSSchedTaskPtr, OSSchedTaskCore,
                 OSSchedTaskPrio, OSCfg_SchedTaskStkBasePtr, 64,
                 OSCfg_SchedTaskFpBasePtr)
    OSAssignTask(OSSchedTaskTCBPtr, CORE1)
    OSInternalTaskPtr[CORE1]    = OSSchedTaskTCBPtr
    OSInternalTaskPrio[CORE1]   = OSSchedTaskPrio

    OSDaemonTaskTCBPtr          = &OS_TCB_DaemonTaskTCB[0]
    OSDaemonTaskPtr             = &OS_DaemonTask
    OSDaemonTaskPrio            = OS_CFG_PRIO_MAX - 1
    OSDaemonTaskCore            = CORE2
    OSCfg_DaemonTaskStkBasePtr  = &OSCfg_DaemonTaskStk[0]
    OSCfg_DaemonTaskFpBasePtr   = &OSCfg_DaemonTaskFp[0]
    OSTaskCreate(OSDaemonTaskTCBPtr, OSDaemonTaskPtr, OSDaemonTaskCore,
                 OSDaemonTaskPrio, OSCfg_DaemonTaskStkBasePtr, 64,
                 OSCfg_DaemonTaskFpBasePtr)
    OSAssignTask(OSDaemonTaskTCBPtr, CORE2)
    OSInternalTaskPtr[CORE2]    = OSDaemonTaskTCBPtr
    OSInternalTaskPrio[CORE2]   = OSDaemonTaskPrio

    OSTask1TCBPtr               = &OS_TCB_Task1TCB[0]
    OS_Task1Ptr                 = &Task1Start
```

```
    OSCfg_Task1StkBasePtr        = &OSCfg_Task1Stk[0]
    OSCfg_Task1FpBasePtr         = &OSCfg_Task1Fp[0]
    OSTaskCreate(OSTask1TCBPtr, OS_Task1Ptr, NO_CORE, 2,
                 OSCfg_Task1StkBasePtr, 64, OSCfg_Task1FpBasePtr)
```

ตารางที่ ค.4 ตัวอย่างชุดรหัสข้อมูลส่วนการจัดการ Semaphore

```
OS_PendingListInsertPrio(p_pend_list, p_pend_data)
    p_tcb = p_pend_data[OS_PD_TCBPtr]
    prio  = p_tcb[OS_TCB_Prio]
    if (p_pend_list[OS_PL_NbrEntries] == 0)
        p_pend_list[OS_PL_NbrEntries] = 1
        p_pend_data[OS_PD_NextPtr]    = 0
        p_pend_data[OS_PD_PrevPtr]    = 0
        p_pend_list[OS_PL_HeadPtr]    = p_pend_data
        p_pend_list[OS_PL_TailPtr]    = p_pend_data
    else
        p_pend_list[OS_PL_NbrEntries] = p_pend_list[OS_PL_NbrEntries] + 1
        p_pend_data_next              = p_pend_list[OS_PL_HeadPtr]
        break_check                   = 1
        while (p_pend_data_next != 0 && break_check == 1)
            p_tcb_next                = p_pend_data_next[OS_PD_TCBPtr]
            if (prio < p_tcb_next[OS_TCB_Prio])
                break_check           = 0
            else
                p_pend_data_next      = p_pend_data_next[OS_PD_NextPtr]
        if (p_pend_data_next == 0)
            p_pend_data[OS_PD_NextPtr]     = 0
            p_pend_data_prev               = p_pend_list[OS_PL_TailPtr]
            p_pend_data[OS_PD_PrevPtr]      = p_pend_data_prev
            p_pend_data_prev[OS_PD_NextPtr] = p_pend_data
            p_pend_list[OS_PL_TailPtr]      = p_pend_data
        else
            if (p_pend_data_next[OS_PD_PrevPtr] == 0)
                p_pend_data_next[OS_PD_PrevPtr]  = p_pend_data
                p_pend_data[OS_PD_PrevPtr]       = 0
                p_pend_data[OS_PD_NextPtr]       = p_pend_data_next
                p_pend_list[OS_PL_HeadPtr]       = p_pend_data
            else
                p_pend_data_prev          = p_pend_data_next[OS_PD_PrevPtr
                p_pend_data[OS_PD_PrevPtr]       = p_pend_data_prev
                p_pend_data[OS_PD_NextPtr]       = p_pend_data_next
                p_pend_data_prev[OS_PD_NextPtr]  = p_pend_data
                p_pend_data_next[OS_PD_PrevPtr]  = p_pend_data

OS_PendingListRemove1(p_pend_list, p_pend_data)
    if (p_pend_list[OS_PL_NbrEntries] == 1)
        p_pend_list[OS_PL_HeadPtr] = 0
        p_pend_list[OS_PL_TailPtr] = 0
    else if (p_pend_data[OS_PD_PrevPtr] == 0)
        p_next                     = p_pend_data[OS_PD_NextPtr]
        p_next[OS_Pd_PrevPtr]      = 0
        p_pend_list[OS_PL_HeadPtr] = p_next
    else if (p_pend_data[OS_PD_NextPtr] == 0)
        p_prev                     = p_pend_data[OS_PD_PrevPtr]
        p_prev[OS_PD_NextPtr]      = 0
        p_pend_list[OS_PL_TailPtr] = p_prev
    else
        p_prev                     = p_pend_data[OS_PD_PrevPtr]
        p_next                     = p_pend_data[OS_PD_NextPtr]
```

```
        p_prev[OS_PD_NextPtr]       = p_next
        p_next[OS_PD_PrevPtr]       = p_prev
    p_pend_list[OS_PL_NbrEntries]   = p_pend_list[OS_PL_NbrEntries] - 1
    p_pend_data[OS_PD_NextPtr]      = 0
    p_pend_data[OS_PD_PrevPtr]      = 0

OS_PendingListRemove(p_tcb)
    p_pend_data = p_tcb[OS_TCB_PendingDataTblPtr]
    n_pend_list = p_tcb[OS_TCB_PendingDataTblEntries]
    if ( n_pend_list > 0 )
        p_sem       = p_pend_data[OS_PD_PendingObjPtr]
        p_pend_list = p_sem[OS_SEM_PendingList]
        OS_PendingListRemove1(p_pend_list, p_pend_data)
        p_tcb[OS_TCB_PendingDataTblEntries] =
p_tcb[OS_TCB_PendingDataTblEntries]
                                          - 1

OS_Pend(p_pend_data, p_sem, pending_on, core)
    p_tcb = OSTCBCurPtr[core]
    p_tcb[OS_TCB_PendingOn]              = pending_on
    p_tcb[OS_TCB_PendingStatus]          = OS_STATUS_PEND_OK
    if (OSTCBCurPtr[core] != OSInternalTaskPtr[core])
        p_rdy                            = OSRdyList[core]
        p_rdy_list                       = p_rdy[OSPrioHighRdy[core]]
        p_tcb[OS_TCB_TaskState]          = OS_TASK_STATE_PEND
        OS_RdyListRemove(p_tcb, p_rdy_list)
    p_pend_list                          = p_sem[OS_SEM_PendingList]
    p_pend_data[OS_PD_PendingObjPtr]     = &p_sem[0]
    p_tcb[OS_TCB_PendingDataTblEntries]  = 1
    p_tcb[OS_TCB_PendingDataTblPtr]      = p_pend_data
    p_pend_data[OS_PD_TCBPtr]            = p_tcb
    OS_PendingListInsertPrio(p_pend_list, p_pend_data)

OSSemPendInInternalTask(p_sem, p_pend_data, core)
    if (core == CORE1)
        asm("trap r21 #18 ; CORE2_DIS")
    else if (core == CORE2)
        asm("trap r21 #24 ; CORE1_DIS")
    p_sem[OS_SEM_Ctr] =  p_sem[OS_SEM_Ctr] - 1
    sem_ctr = p_sem[OS_SEM_Ctr]
    if (core == CORE1)
        asm("trap r21 #19 ; CORE2_EN")
    else if (core == CORE2)
        asm("trap r21 #25 ; CORE1_EN")
    while(sem_ctr < 0)
        if (core == CORE1)
            asm("trap r21 #20 ; CORE1_INT_DIS")
        else if (core == CORE2)
            asm("trap r21 #22 ; CORE2_INT_DIS")
        p_tbl                        = OSPrioTbl[core]
        OSPrioHighRdy[core] = OS_PrioGetHighest(OSTCBCurPtr[core], p_tbl)
        p_rdy                        = OSRdyList[core]
        p_rdy_list                   = p_rdy[OSPrioHighRdy[core]]
        p_tcb                        = p_rdy_list[OS_Rdy_HeadPtr]
        OSTCBHighRdyPtr[core]        = p_tcb
        if ( OSTCBHighRdyPtr[core] != OSInternalTaskPtr[core] &&
             p_tcb[OS_TCB_TaskState] == OS_TASK_STATE_RDY)
            if (core == CORE1)
                asm("trap r21 #21 ; CORE1_INT_EN")
                asm("trap r21 #16 ; SW_INT_CORE1")
            else if (core == CORE2)
                asm("trap r21 #23 ; CORE2_INT_EN")
                asm("trap r21 #17 ; SW_INT_CORE2")
        if (core == CORE1)
```

```
            asm("trap r21 #18 ; CORE2_DIS")
        else if (core == CORE2)
            asm("trap r21 #24 ; CORE1_DIS")
        sem_ctr = p_sem[OS_SEM_Ctr]
        if (core == CORE1)
            asm("trap r21 #19 ; CORE2_EN")
        else if (core == CORE2)
            asm("trap r21 #25 ; CORE1_EN")
        if (core == CORE1)
            asm("trap r21 #21 ; CORE1_INT_EN")
        else if (core == CORE2)
            asm("trap r21 #23 ; CORE2_INT_EN")

OSSemPend(p_sem, p_pend_data, core)
    if (core == CORE1)
        asm("trap r21 #20 ; CORE1_INT_DIS")
    else if (core == CORE2)
        asm("trap r21 #22 ; CORE2_INT_DIS")
    sem_ctr = 0
    if (OSTCBCurPtr[core] == OSInternalTaskPtr[core])
        OSSemPendInInternalTask(p_sem, p_pend_data, core)
        return
    else
        if (core == CORE1)
            asm("trap r21 #18 ; CORE2_DIS")
        else if (core == CORE2)
            asm("trap r21 #24 ; CORE1_DIS")
        sem_ctr = p_sem[OS_SEM_Ctr]
        if (core == CORE1)
            asm("trap r21 #19 ; CORE2_EN")
        else if (core == CORE2)
            asm("trap r21 #25 ; CORE1_EN")
        if (sem_ctr > 0)    // Resource available?
            if (core == CORE1)
                asm("trap r21 #18 ; CORE2_DIS")
            else if (core == CORE2)
                asm("trap r21 #24 ; CORE1_DIS")
            p_sem[OS_SEM_Ctr] =  p_sem[OS_SEM_Ctr] - 1
            if (core == CORE1)
                asm("trap r21 #19 ; CORE2_EN")
            else if (core == CORE2)
                asm("trap r21 #25 ; CORE1_EN")
            if (core == CORE1)
                asm("trap r21 #21 ; CORE1_INT_EN")
            else if (core == CORE2)
                asm("trap r21 #23 ; CORE2_INT_EN")
            return
        if (core == CORE1)
            asm("trap r21 #19 ; CORE2_EN")
        else if (core == CORE2)
            asm("trap r21 #25 ; CORE1_EN")
        OS_Pend(p_pend_data, p_sem, OS_TASK_PEND_ON_SEM, core)
        OSTCBHighRdyPtr[core]   = OSInternalTaskPtr[core]
        OSPrioHighRdy[core]     = OSInternalTaskPrio[core]
        if (core == CORE1)
            asm("trap r21 #18 ; CORE2_DIS")
        else if (core == CORE2)
            asm("trap r21 #24 ; CORE1_DIS")
        p_sem[OS_SEM_Ctr] =  p_sem[OS_SEM_Ctr] - 1
        if (core == CORE1)
            asm("trap r21 #19 ; CORE2_EN")
        else if (core == CORE2)
            asm("trap r21 #25 ; CORE1_EN")
        if (core == CORE1)
```

```
            asm("trap r21 #21 ; CORE1_INT_EN")
            asm("trap r21 #16 ; SW_INT_CORE1")
        else if (core == CORE2)
            asm("trap r21 #23 ; CORE2_INT_EN")
            asm("trap r21 #17 ; SW_INT_CORE2")
    if (core == CORE1)
        asm("trap r21 #21 ; CORE1_INT_EN")
    else if (core == CORE2)
        asm("trap r21 #23 ; CORE2_INT_EN")
    return

OS_Post(p_sem, p_tcb, core)
    OS_PendingListRemove(p_tcb)
    p_rdy                       = OSRdyList[core]
    p_rdy_list                  = p_rdy[OSPrioHighRdy[core]]
    os_prio_cur                 = p_tcb[OS_TCB_Prio]
    if (p_tcb[OS_TCB_TaskState] == OS_TASK_STATE_PEND)
        OS_RdyListInsert(p_tcb, p_rdy_list, os_prio_cur)
    p_tcb[OS_TCB_TaskState]     = OS_TASK_STATE_RDY
    p_tcb[OS_TCB_PendingStatus]    = OS_STATUS_PEND_OK
    p_tcb[OS_TCB_PendingOn]        = OS_TASK_PEND_ON_NOTHING

OS_SemPostDiffCore(p_sem, p_tcb, cur_core, post_core)
    if (cur_core == CORE1)
        asm("trap r21 #18 ; CORE2_DIS")
    else if (cur_core == CORE2)
        asm("trap r21 #24 ; CORE1_DIS")
    OS_Post(p_sem, p_tcb, post_core)
    if (cur_core == CORE1)
        asm("trap r21 #19 ; CORE2_EN")
    else if (cur_core == CORE2)
        asm("trap r21 #25 ; CORE1_EN")
    p_pend_list = 0
    if (p_tcb == OSInternalTaskPtr[post_core])
        p_pend_list = p_sem[OS_SEM_PendingList]
        if (p_pend_list[OS_PL_NbrEntries] == 0)
            if (cur_core == CORE1)
                asm("trap r21 #18 ; CORE2_DIS")
            else if (cur_core == CORE2)
                asm("trap r21 #24 ; CORE1_DIS")
            if (p_sem[OS_SEM_Ctr] < 1)
                p_sem[OS_SEM_Ctr] = p_sem[OS_SEM_Ctr] + 1
            if (cur_core == CORE1)
                asm("trap r21 #19 ; CORE2_EN")
            else if (cur_core == CORE2)
                asm("trap r21 #25 ; CORE1_EN")

OS_SemPostSameCore(p_sem, p_tcb, cur_core, post_core)
    OS_Post(p_sem, p_tcb, post_core)
    if (OSTCBCurPtr[post_core] == OSInternalTaskPtr[post_core])
        OSTCBHighRdyPtr[post_core]   = p_tcb
        OSPrioHighRdy[post_core]     = p_tcb[OS_TCB_Prio]
        if (post_core == CORE1)
            asm("trap r21 #21 ; CORE1_INT_EN")
            asm("trap r21 #16 ; SW_INT_CORE1")
        else if (post_core == CORE2)
            asm("trap r21 #23 ; CORE2_INT_EN")
            asm("trap r21 #17 ; SW_INT_CORE2")
    else
        OSTCBHighRdyPtr[post_core]   = OSInternalTaskPtr[post_core]
        OSPrioHighRdy[post_core]     = OSInternalTaskPrio[post_core]
        if (post_core == CORE1)
            asm("trap r21 #21 ; CORE1_INT_EN")
            asm("trap r21 #16 ; SW_INT_CORE1")
```

```
        else if (post_core == CORE2)
            asm("trap r21 #21 ; CORE1_INT_EN")
            asm("trap r21 #17 ; SW_INT_CORE2")

OS_SemPostInInternalTask(p_sem, p_pend_data, core)
    p_pend_list = p_sem[OS_SEM_PendingList]
    if (p_pend_list[OS_PL_NbrEntries] == 0)
        return
    p_tcb = p_pend_data[OS_PD_TCBPtr]
    post_core = p_tcb[OS_TCB_CoreRunning]
    if ( core == post_core )                // Post in the same core
        OS_SemPostSameCore(p_sem, p_tcb, core, post_core)
    else                                    // Post in the different core
        OS_SemPostDiffCore(p_sem, p_tcb, core, post_core)

OS_SemPost(p_sem, p_pend_data, core)
    if (core == CORE1)
        asm("trap r21 #20 ; CORE1_INT_DIS")
    else if (core == CORE2)
        asm("trap r21 #22 ; CORE2_INT_DIS")
    if (core == CORE1)
        asm("trap r21 #18 ; CORE2_DIS")
    else if (core == CORE2)
        asm("trap r21 #24 ; CORE1_DIS")
    if (p_sem[OS_SEM_Ctr] < 1)
        p_sem[OS_SEM_Ctr] =  p_sem[OS_SEM_Ctr] + 1
    if (core == CORE1)
        asm("trap r21 #19 ; CORE2_EN")
    else if (core == CORE2)
        asm("trap r21 #25 ; CORE1_EN")
    if (OSTCBCurPtr[core] == OSInternalTaskPtr[core])
        OS_SemPostInInternalTask(p_sem, p_pend_data, core)
    else
        p_pend_list = p_sem[OS_SEM_PendingList]
        if (p_pend_list[OS_PL_NbrEntries] == 0)
            if (core == CORE1)
                asm("trap r21 #21 ; CORE1_INT_EN")
            else if (core == CORE2)
                asm("trap r21 #23 ; CORE2_INT_EN")
            return

        p_tcb = p_pend_data[OS_PD_TCBPtr]
        post_core = p_tcb[OS_TCB_CoreRunning]
        if ( core == post_core )  // Post in the same core
            OS_SemPostSameCore(p_sem, p_tcb, core, post_core)
        else                      // Post in the different core
            OS_SemPostDiffCore(p_sem, p_tcb, core, post_core)
        if (core == CORE1)
            asm("trap r21 #21 ; CORE1_INT_EN")
        else if (core == CORE2)
            asm("trap r21 #23 ; CORE2_INT_EN")
```

ตารางที่ ค.5 ตัวอย่างชุดรหัสข้อมูลส่วนการจัดการตารางงาน

```
OS_SchedRoundRobin(p_rdy_list)
    p_tcb = p_rdy_list[OS_Rdy_HeadPtr]
    if (p_tcb == 0)
        return
    if (p_tcb[OS_TCB_TimeQuantaCtr] > 0)
        p_tcb[OS_TCB_TimeQuantaCtr] = p_tcb[OS_TCB_TimeQuantaCtr] - 1
```

```
    if (p_tcb[OS_TCB_TimeQuantaCtr] > 0)
        return
    if (p_rdy_list[OS_Rdy_NbrEntries] < 2)
        return
    OS_RdyListMoveHeadToTail(p_rdy_list)
    p_tcb = p_rdy_list[OS_Rdy_HeadPtr]
    if (p_tcb[OS_TCB_TimeQuanta] == 0)
        p_tcb[OS_TCB_TimeQuantaCtr] = OSSchedRoundRobinDfltTimeQuanta;
    else
        p_tcb[OS_TCB_TimeQuantaCtr] = p_tcb[OS_TCB_TimeQuanta]

OS_SchedTask()
    asm("mov r1 #interrupt ; set Timer1 vector")
    asm("st r1 10")
    asm("mov r1 #interrupt2 ; set Timer2 vector")
    asm("st r1 11")
    asm("mov r1 #interrupt3 ; set SW1 vector")
    asm("st r1 12")
    asm("mov r1 #interrupt4 ; set SW2 vector")
    asm("st r1 13")
    asm("mov r1 #interrupt5 ; set Print End")
    asm("st r1 14")
    asm("mov r1 #1")
    asm("trap r1 #11  ; set div = 1")
    asm("mov r1 #10000")
    asm("trap r1 #10 ; set timer1 = 10000")
    asm("trap r1 #13 ; enable interrupt")

    while (1)
        asm("trap r21 #20 ; CORE1_INT_DIS")
        p_tbl                       = OSPrioTbl[CORE1]
        OSPrioHighRdy[CORE1] = OS_PrioGetHighest(OSTCBCurPtr[CORE1], p_tbl)
        p_rdy                       = OSRdyList[CORE1]
        p_rdy_list                  = p_rdy[OSPrioHighRdy[CORE1]]
        OS_SchedRoundRobin(p_rdy_list)
        p_tcb = OSEventOccur()
        if ( p_tcb != 0 )
            core = p_tcb[OS_TCB_CoreAffinity]
            if ( core == NO_CORE )
                core = CORE1
                if ( Core2Enable == 0 )
                    core = CORE1
                else if ( OSTaskQty[CORE1] == 0 )
                    core = CORE1
                else if ( OSTaskQty[CORE1] > OSTaskQty[CORE2] )
                    core = CORE2
            if ( core == CORE1 )
                OSAssignTask(p_tcb, core)
                OSPrioHighRdy[CORE1]    =
                                OS_PrioGetHighest(OSTCBCurPtr[CORE1], p_tbl)
                p_rdy_list              = p_rdy[OSPrioHighRdy[CORE1]]
            else
                OSSemPend(SemAssignPtr, SemAssignPendingDataPtr, CORE1)
                if ( ShareAssignSignal == 0 )
                    ShareAssignTCBPtr   = p_tcb
                    ShareAssignSignal   = 1
                OS_SemPost(SemAssignPtr, SemAssignPendingDataPtr, CORE1)
            EventNum                = EventNum + 1
        p_tcb                       = p_rdy_list[OS_Rdy_HeadPtr]
        OSTCBHighRdyPtr[CORE1]      = p_tcb

        if ( OSTCBHighRdyPtr[CORE1] != OSSchedTaskTCBPtr &&
                    p_tcb[OS_TCB_TaskState] ==
             OS_TASK_STATE_RDY)
```

```
            asm("trap r21 #21 ; CORE1_INT_EN")
            asm("trap r21 #16 ; SW_INT_CORE1")
        else
            asm("trap r21 #21 ; CORE1_INT_EN")

OS_DaemonTask()
    while (1)
        asm("trap r21 #22 ; CORE2_INT_DIS")
        p_tbl                       = OSPrioTbl[CORE2]
        OSPrioHighRdy[CORE2] = OS_PrioGetHighest(OSTCBCurPtr[CORE2], p_tbl)
        p_rdy                       = OSRdyList[CORE2]
        p_rdy_list                  = p_rdy[OSPrioHighRdy[CORE2]]
        OS_SchedRoundRobin(p_rdy_list)
        p_tcb                       = 0

        OSSemPend(SemAssignPtr, SemAssignPendingDataPtr, CORE2)
        if ( ShareAssignSignal == 1 )
            p_tcb = ShareAssignTCBPtr
            ShareAssignTCBPtr       = 0
            ShareAssignSignal       = 0
        OS_SemPost(SemAssignPtr, SemAssignPendingDataPtr, CORE2)

        if ( p_tcb != 0 )
            OSAssignTask(p_tcb, CORE2)
        p_tcb                       = p_rdy_list[OS_Rdy_HeadPtr]
        OSTCBHighRdyPtr[CORE2]      = p_tcb

        if ( OSTCBHighRdyPtr[CORE2] != OSDaemonTaskTCBPtr &&
                p_tcb[OS_TCB_TaskState] == OS_TASK_STATE_RDY )
            asm("trap r21 #23 ; CORE2_INT_EN")
            asm("trap r21 #17 ; SW_INT_CORE2")
        else
            asm("trap r21 #23 ; CORE2_INT_EN")
```

ตารางที่ ค.6 ตัวอย่างชุดรหัสข้อมูลส่วนฟังก์ชันสัญญาณขัดจังหวะ

```
; Timer1 Interrupt Core 1
:interrupt
; CORE1_INT_DIS
trap r21 #20
; save rads
mov r24 rads
; save retval
mov r25 retval
jal rads OSIntTimer1
mov r21 r28
jf r21 INT1RET
; Save CPU Registers
; r21 = OSTCBCurPtr
mov r21 #0
ld r21 OSTCBCurPtr
mov r23 #9
ld r22 +r21 r23
jf r22 INT1NS
; save sp
st sp @0 r21
; save fp
st fp @1 r21
; save rads
st r24 @18 r21
```

```
; save ads
st r31 @10 r21
; r21 = OSTCBCurPtr[OS_TCB_StkBasePtr]
mov r22 #9
ld r21 +r21 r22
; save r1 - r20
st r1 @1 r21
st r2 @2 r21
st r3 @3 r21
st r4 @4 r21
st r5 @5 r21
st r6 @6 r21
st r7 @7 r21
st r8 @8 r21
st r9 @9 r21
st r10 @10 r21
st r11 @11 r21
st r12 @12 r21
st r13 @13 r21
st r14 @14 r21
st r15 @15 r21
st r16 @16 r21
st r17 @17 r21
st r18 @18 r21
st r19 @19 r21
st r20 @20 r21
; save retval=r28
st r25 @28 r21
:INT1NS
; OSPrioCur = OSPrioHighRdy
mov r21 #0
ld r21 OSPrioHighRdy
st r21 OSPrioCur
; OSTCBCurPtr = OSTCBHighRdyPtr
mov r21 #0
ld r21 OSTCBHighRdyPtr
st r21 OSTCBCurPtr
; Restore CPU Registers
; r21 = OSTCBCurPtr
mov r21 #0
ld r21 OSTCBCurPtr
; load sp
ld sp @0 r21
; load fp
ld fp @1 r21
; load rads
ld rads @18 r21
; load ads
ld r31 @10 r21
; r21 = OSTCBCurPtr[OS_TCB_StkBasePtr]
mov r22 #9
ld r21 +r21 r22
; load r1 - 20
ld r1 @1 r21
ld r2 @2 r21
ld r3 @3 r21
ld r4 @4 r21
ld r5 @5 r21
ld r6 @6 r21
ld r7 @7 r21
ld r8 @8 r21
ld r9 @9 r21
ld r10 @10 r21
ld r11 @11 r21
```

```
ld r12 @12 r21
ld r13 @13 r21
ld r14 @14 r21
ld r15 @15 r21
ld r16 @16 r21
ld r17 @17 r21
ld r18 @18 r21
ld r19 @19 r21
ld r20 @20 r21
; load retval=r28
ld r28 @28 r21
:INT1RET
; CORE1_INT_EN
trap r21 #21
ret r31
; Timer2 Interrupt Core 2
:interrupt2
; CORE2_INT_DIS
trap r21 #22
; save rads
mov r24 rads
; save retval
mov r25 retval
jal rads OSIntTimer2
mov r21 r28
jf r21 INT2RET
; Save CPU Registers
; r21 = OSTCBCurPtr[1]
mov r22 #1
ld r21 @OSTCBCurPtr r22
mov r23 #9
ld r22 +r21 r23
jf r22 INT2NS
; save sp
st sp @0 r21
; save fp
st fp @1 r21
; save rads
st r24 @18 r21
; save ads
st r31 @10 r21
; r21 = OSTCBCurPtr[OS_TCB_StkBasePtr]
mov r22 #9
ld r21 +r21 r22
; save r1 - r20
st r1 @1 r21
st r2 @2 r21
st r3 @3 r21
st r4 @4 r21
st r5 @5 r21
st r6 @6 r21
st r7 @7 r21
st r8 @8 r21
st r9 @9 r21
st r10 @10 r21
st r11 @11 r21
st r12 @12 r21
st r13 @13 r21
st r14 @14 r21
st r15 @15 r21
st r16 @16 r21
st r17 @17 r21
st r18 @18 r21
st r19 @19 r21
```

```
st r20 @20 r21
; save retval=r28
st r25 @28 r21
:INT2NS
; OSPrioCur[1] = OSPrioHighRdy[1]
mov r22 #1
ld r21 @OSPrioHighRdy r22
st r21 @OSPrioCur r22
; OSTCBCurPtr[1] = OSTCBHighRdyPtr[1]
mov r22 #1
ld r21 @OSTCBHighRdyPtr r22
st r21 @OSTCBCurPtr r22
; Restore CPU Registers
; r21 = OSTCBCurPtr[1]
mov r22 #1
ld r21 @OSTCBCurPtr r22
; load sp
ld sp @0 r21
; load fp
ld fp @1 r21
; load rads
ld rads @18 r21
; load ads
ld r31 @10 r21
; r21 = OSTCBCurPtr[OS_TCB_StkBasePtr]
mov r22 #9
ld r21 +r21 r22
; load r1 - 20
ld r1 @1 r21
ld r2 @2 r21
ld r3 @3 r21
ld r4 @4 r21
ld r5 @5 r21
ld r6 @6 r21
ld r7 @7 r21
ld r8 @8 r21
ld r9 @9 r21
ld r10 @10 r21
ld r11 @11 r21
ld r12 @12 r21
ld r13 @13 r21
ld r14 @14 r21
ld r15 @15 r21
ld r16 @16 r21
ld r17 @17 r21
ld r18 @18 r21
ld r19 @19 r21
ld r20 @20 r21
; load retval=r28
ld r28 @28 r21
:INT2RET
; CORE2_INT_EN
trap r21 #23
ret r31
; SW1 Core1
:interrupt3
; CORE1_INT_DIS
trap r21 #20
; save rads
mov r24 rads
; save retval
mov r25 retval
; Save CPU Registers
; r21 = OSTCBCurPtr
```

```
mov r21 #0
ld r21 OSTCBCurPtr
mov r23 #9
ld r22 +r21 r23
jf r22 INT3NS
; save sp
st sp @0 r21
; save fp
st fp @1 r21
; save rads
st r24 @18 r21
; save ads
st r31 @10 r21
; r21 = OSTCBCurPtr[OS_TCB_StkBasePtr]
mov r22 #9
ld r21 +r21 r22
; save r1 - r20
st r1 @1 r21
st r2 @2 r21
st r3 @3 r21
st r4 @4 r21
st r5 @5 r21
st r6 @6 r21
st r7 @7 r21
st r8 @8 r21
st r9 @9 r21
st r10 @10 r21
st r11 @11 r21
st r12 @12 r21
st r13 @13 r21
st r14 @14 r21
st r15 @15 r21
st r16 @16 r21
st r17 @17 r21
st r18 @18 r21
st r19 @19 r21
st r20 @20 r21
; save retval=r28
st r25 @28 r21
:INT3NS
; OSPrioCur = OSPrioHighRdy
mov r21 #0
ld r21 OSPrioHighRdy
st r21 OSPrioCur
; OSTCBCurPtr = OSTCBHighRdyPtr
mov r21 #0
ld r21 OSTCBHighRdyPtr
st r21 OSTCBCurPtr
; Restore CPU Registers
; r21 = OSTCBCurPtr
mov r21 #0
ld r21 OSTCBCurPtr
; load sp
ld sp @0 r21
; load fp
ld fp @1 r21
; load rads
ld rads @18 r21
; load ads
ld r31 @10 r21
; r21 = OSTCBCurPtr[OS_TCB_StkBasePtr]
mov r22 #9
ld r21 +r21 r22
; load r1 - r20
```

```
ld r1 @1 r21
ld r2 @2 r21
ld r3 @3 r21
ld r4 @4 r21
ld r5 @5 r21
ld r6 @6 r21
ld r7 @7 r21
ld r8 @8 r21
ld r9 @9 r21
ld r10 @10 r21
ld r11 @11 r21
ld r12 @12 r21
ld r13 @13 r21
ld r14 @14 r21
ld r15 @15 r21
ld r16 @16 r21
ld r17 @17 r21
ld r18 @18 r21
ld r19 @19 r21
ld r20 @20 r21
; load retval=r28
ld r28 @28 r21
; CORE1_INT_EN
trap r21 #21
ret r31
; SW1 Core2
:interrupt4
; CORE2_INT_DIS
trap r21 #22
; save rads
mov r24 rads
; save retval
mov r25 retval
; Save CPU Registers
; r21 = OSTCBCurPtr[1]
mov r22 #1
ld r21 @OSTCBCurPtr r22
mov r23 #9
ld r22 +r21 r23
jf r22 INT4NS
; save sp
st sp @0 r21
; save fp
st fp @1 r21
; save rads
st r24 @18 r21
; save ads
st r31 @10 r21
; r21 = OSTCBCurPtr[OS_TCB_StkBasePtr]
mov r22 #9
ld r21 +r21 r22
; save r1 - r20
st r1 @1 r21
st r2 @2 r21
st r3 @3 r21
st r4 @4 r21
st r5 @5 r21
st r6 @6 r21
st r7 @7 r21
st r8 @8 r21
st r9 @9 r21
st r10 @10 r21
st r11 @11 r21
st r12 @12 r21
```

```
st r13 @13 r21
st r14 @14 r21
st r15 @15 r21
st r16 @16 r21
st r17 @17 r21
st r18 @18 r21
st r19 @19 r21
st r20 @20 r21
; save retval=r28
st r25 @28 r21
:INT4NS
; OSPrioCur[1] = OSPrioHighRdy[1]
mov r22 #1
ld r21 @OSPrioHighRdy r22
st r21 @OSPrioCur r22
; OSTCBCurPtr[1] = OSTCBHighRdyPtr[1]
mov r22 #1
ld r21 @OSTCBHighRdyPtr r22
st r21 @OSTCBCurPtr r22
; Restore CPU Registers
; r21 = OSTCBCurPtr[1]
mov r22 #1
ld r21 @OSTCBCurPtr r22
; load sp
ld sp @0 r21
; load fp
ld fp @1 r21
; load rads
ld rads @18 r21
; load ads
ld r31 @10 r21
; r21 = OSTCBCurPtr[OS_TCB_StkBasePtr]
mov r22 #9
ld r21 +r21 r22
; load r1 - r20
ld r1 @1 r21
ld r2 @2 r21
ld r3 @3 r21
ld r4 @4 r21
ld r5 @5 r21
ld r6 @6 r21
ld r7 @7 r21
ld r8 @8 r21
ld r9 @9 r21
ld r10 @10 r21
ld r11 @11 r21
ld r12 @12 r21
ld r13 @13 r21
ld r14 @14 r21
ld r15 @15 r21
ld r16 @16 r21
ld r17 @17 r21
ld r18 @18 r21
ld r19 @19 r21
ld r20 @20 r21
; load retval=r28
ld r28 @28 r21
; CORE2_INT_EN
trap r21 #23
ret r31
```

ตารางที่ ค.7 ตัวอย่างชุดรหัสข้อมูลเริ่มการทำงาน

```
OSInit()
    Core2Enable                          = 1
    OSRunning                            = 0
    OSSchedRoundRobinDfltTimeQuanta      = 2
    SemAssign[OS_SEM_Ctr]                = 1
    SemAssign[OS_SEM_TS]                 = 0
    SemAssign[OS_SEM_PendingList]        = &SemAssignPendingList[0]
    SemAssignPendingDataPtr              = &SemAssignPendingData[0]
    SemAssignPtr                         = &SemAssign[0]
    OSPrioTbl[0]                         = &OSPrio1Tbl[0]
    OSPrioTbl[1]                         = &OSPrio2Tbl[0]
    OSRdy1List[0]                        = &OSRdy1List0[0]
    OSRdy1List[1]                        = &OSRdy1List1[0]
    OSRdy1List[2]                        = &OSRdy1List2[0]
    OSRdy1List[3]                        = &OSRdy1List3[0]
    OSRdy1List[4]                        = &OSRdy1List4[0]
    OSRdy1List[5]                        = &OSRdy1List5[0]
    OSRdy1List[6]                        = &OSRdy1List6[0]
    OSRdy1List[7]                        = &OSRdy1List7[0]
    OSRdyList[0]                         = &OSRdy1List[0]
    OSRdy2List[0]                        = &OSRdy2List0[0]
    OSRdy2List[1]                        = &OSRdy2List1[0]
    OSRdy2List[2]                        = &OSRdy2List2[0]
    OSRdy2List[3]                        = &OSRdy2List3[0]
    OSRdy2List[4]                        = &OSRdy2List4[0]
    OSRdy2List[5]                        = &OSRdy2List5[0]
    OSRdy2List[6]                        = &OSRdy2List6[0]
    OSRdy2List[7]                        = &OSRdy2List7[0]
    OSRdyList[1]                         = &OSRdy2List[0]
    OSTaskQty[0]                         = 0 // Clear the number of tasks
    OSTaskQty[1]                         = 0 // Clear the number of tasks
    CreateTask()
    SimEventInit()

OSStartHighRdy()
    asm("ld r21 OSTCBCurPtr")
    asm("ld sp @0 r21")
    asm("ld fp @1 r21")
    asm("ld r22 @10 r21")
    asm("ret r22")

OSStart()
    if ( OSRunning == 0 )
        OSTCBHighRdyPtr[CORE1]       = &OS_TCB_SchedTaskTCB[0]
        OSTCBCurPtr[CORE1]           = &OS_TCB_SchedTaskTCB[0]
        OSPrioHighRdy[CORE1]         = OS_CFG_PRIO_MAX - 1
        OSPrioCur[CORE1]             = OS_CFG_PRIO_MAX - 1
        OSTCBHighRdyPtr[CORE2]       = &OS_TCB_DaemonTaskTCB[0]
        OSTCBCurPtr[CORE2]           = &OS_TCB_DaemonTaskTCB[0]
        OSPrioHighRdy[CORE2]         = OS_CFG_PRIO_MAX - 1
        OSPrioCur[CORE2]             = OS_CFG_PRIO_MAX - 1
        OSRunning                    = OS_STATE_OS_RUNNING
        StartTime                    = OS_TS_GET()

        if ( Core2Enable )
            asm("trap r22 #15")      // start core 2
            i = 0
            while ( i < 20 )         // wait core 2 start
                i = i + 1
        OSStartHighRdy()
```

```
OSStartCore2()
    asm("mov r22 #1")
    asm("ld r21 @OSTCBCurPtr r22")
    asm("ld sp @0 r21")
    asm("ld fp @1 r21")
    asm("ld r22 @10 r21")
    asm("ret r22")

main2()
    OSStartCore2()

main()
    OSInit()
    OSStart()
```

ประวัติผู้เขียนวิทยานิพนธ์

นายอังคาร เชี่ยวกิจวุฒิกุล เกิดเมื่อวันที่ 15 กันยายน 2524 ที่ตำบลฉวาง อำเภอฉวาง จังหวัดนครศรีธรรมราช สำเร็จการศึกษาปริญญาวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์ เมื่อปีการศึกษา 2547 และเข้าศึกษาในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2554

ผู้เขียนวิทยานิพนธ์ได้ตีพิมพ์ผลงานทางวิชาการในรายงานการประชุม “The 9th National Conference on Computing and Information Technology, NCCIT2013” ชื่อว่า “Scheduling Tasks in Real-time Operating Systems on Multiple Core Microprocessors” ณ คณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ (KMUTNB) ประเทศไทย เมื่อเดือนพฤษภาคม พ.ศ. 2556