บทที่ 2
การใช้งานซอฟท์แวร์ในระบบเวลาจริง

2.1 ระบบปฏิบัติการแบบเวลาจริง

ระบบเวลาจริง [2] หมายถึง ระบบซึ่งความถูกต้องไม่ขึ้นกับผลลัพธ์ในการคำนวณของคอมพิวเตอร์เพียงอย่างเดียว ยังขึ้นกับเวลาที่ผลลัพธ์ถูกสร้างขึ้นอีกด้วย นั่นคือข้อจำกัดทางเวลาจะถูกกำหนดให้กับกระบวนการหรืองานของระบบ ในการที่ระบบจะทำงานได้ตามข้อกำหนดทางเวลาเหล่านี้ ระบบต้องการสิ่งที่เรียกว่า การทำนายได้ (predictability) ซึ่งในแง่ของระบบปฏิบัติการจะหมายถึงความเร็วในการตอบสนองต่อการอินเทอร์รัพและการจัดลำดับงานจะต้องอยู่ในเกณฑ์ที่ยอมรับได้

ข้อจำกัดทางเวลาหมายถึง การที่จะรับประกันว่างานจะเสร็จภายในขอบเขตเวลาที่กำหนด ตัวอย่างของงานที่มีข้อจำกัดทางเวลาได้แก่ การควบคุมหุ่นยนต์ และโรงไฟฟ้าเป็นต้น ซึ่งในสถานการณ์เหล่านี้ ข้อมูลที่ได้รับจากสิ่งแวดล้อมจะต้องสอดคล้องกับสภาวะจริงของระบบ มิฉะนั้นอาจก่อให้เกิดความเสียหายได้ ข้อจำกัดทางเวลามีทั้งแบบเป็นคาบเวลา (periodic) ซึ่งหมายความว่า งานจะต้องถูกทำทุก ๆ ช่วงเวลาที่แน่นอน หรือแบบไม่เป็นคาบเวลา (aperiodic) ซึ่งงานจะต้องถูกทำให้เสร็จภายในเวลาที่กำหนดซึ่งไม่แน่นอน หรืออาจเป็นทั้งสองแบบรวมกันก็ได้

งานแบบเวลาจริง แบ่งได้เป็นแบบเข้มงวด (hard real-time) ซึ่งงานจะต้องเริ่มทำและหรือถูกทำให้เสร็จก่อนภายในขอบเขตเวลาที่กำหนด ไม่สามารถผิดพลาดได้ และแบบไม่เข้มงวด (soft real-time) ซึ่งการทำงานอาจผิดพลาดจากขอบเขตเวลาไปบ้างเป็นบางครั้ง แต่ไม่มากนัก

การพัฒนาตัวควบคุมโดยตรงต้องการการทำงานเวลาจริงแบบเข้มงวด (hard real-time) เพื่อรับประกันว่าข้อมูลต่าง ๆ จะถูกต้องตรงกับสภาวะจริงของระบบ และสัญญาณควบคุมจะถูกส่งไปอย่างทันท่วงที และไม่สามารถละเลยหรือผิดพลาดได้

ปัจจัยที่ต้องคำนึงถึงในการพิจารณาระบบปฏิบัติการแบบเวลาจริง [9,10] ได้แก่ การจัดลำดับการทำงาน การจัดโครงสร้างความสำคัญ การจัดการงาน การจัดการหน่วยความจำ การติดต่อระหว่างงาน และการตรวจสอบซึ่งกันและกัน ซึ่งจะได้กล่าวในหัวข้อต่อไป ทั้งนี้เพื่อที่จะรับประกันความถูกต้องของเวลาที่ผลลัพธ์จะเกิดขึ้น นอกจากนี้ในการกล่าวถึงการทำงานแบบเวลาจริง (real-time) ต่อจากนี้ หากมิได้ระบุชัดเจนจะหมายถึงงานเวลาจริงแบบเข้มงวด (hard real-time)

2.2 การจัดลำดับการทำงาน (Scheduling Strategies)

การจัดลำดับการทำงานคือการกำหนดว่าในเวลาใดควรจะทำงานชิ้นใด และงานที่มีความสำคัญต่างกัน ควรจะทำงานชิ้นใดก่อนเป็นเวลาเท่าใด เป็นต้น นอกจากนี้งานแต่ละงานอาจมีข้อกำหนดทางเวลา เช่น เวลาที่เริ่มทำ เวลาที่ต้องทำเสร็จ หรือทำทุก ๆ คาบเวลา อีกด้วย โดยทั่วไปงานจะถูกจัดให้รออยู่ในลักษณะของqueue หรือ Link-list การจัดลำดับการทำงานสามารถทำได้หลายรูปแบบคือ Cyclic, Round-robin [8], Nice level [8], Priority level, Pre-emptive, Rate-monotonic [10], Earliest deadline [10], Least slack [10] โดยมีรายละเอียดดังต่อไปนี้

2.2.1 Cyclic

การจัดการทำงานในรูปแบบ Cyclic มีดังนี้คือ สำหรับงานปัจจุบันตัวจัดการ (Scheduler) จะจองการใช้งานซีพียูจนกว่าจะทำงานเสร็จ จากนั้นก็จะเปลี่ยนไปยังงานถัดไปที่รออยู่ เป็นวิธีที่ง่ายและมีประสิทธิภาพเพราะสามารถลดเวลาในการเปลี่ยนการทำงานจากงานหนึ่งไปยังอีกงานหนึ่งได้ อีกทั้งยังเป็นวิธีที่ใช้ได้ผลดี สำหรับระบบฝังตัว (embedded system) ซึ่งงานแต่ละงานจะถูกออกแบบมาให้มีขนาดเล็กและใช้เวลาค่อนข้างซีพียูคงที่และเท่า ๆ กัน แต่สำหรับระบบโดยทั่วไปเป็นการยากที่จะกำหนดให้งานแต่ละงานมีลักษณะดังกล่าวได้

2.2.2 Round-robin

มีลักษณะคล้ายกับแบบ Cyclic ต่างกันเพียงเล็กน้อยตรงที่ ในแบบ round robin จะมีการกำหนดขีดจำกัดสำหรับการทำงานแต่ละงาน ถ้าหากงานปัจจุบันใช้งานซีพียูนานจนถึงขีดจำกัดแล้ว ตัวจัดการก็จะทำการเปลี่ยนการทำงานไปยังงานถัดไปที่รออยู่และงานเดิมก็จะถูกย้ายต่อท้ายใน link list เพื่อรอเวลาการใช้งานซีพียูต่อไป สำหรับวิธีนี้ จะช่วยให้การทำงานไม่ตกอยู่กับงานใดงานหนึ่งนานเกินกว่าขีดจำกัดที่กำหนด

2.2.3 Nice level

เป็นการจัดลำดับการทำงานที่คำนึงถึงสมรรถนะโดยรวมของระบบ มีลักษณะดังนี้ คือ ตัวจัดการจะถูกเรียกทุก ๆ คาบเวลาหนึ่ง ๆ และในแต่ละครั้งก็จะมีการคำนวณค่าความเหมาะสมในการทำงานสำหรับแต่ละงานที่รออยู่ งานที่เหมาะสมที่สุดก็จะถูกทำก่อน การคำนวณค่าความเหมาะสมขึ้นอยู่กับตัวแปร เช่น ระยะเวลาที่รอ ความสำคัญของเจ้าของงาน เป็นต้น การจัดลำดับงานแบบนี้มีข้อเสียคือ ต้องเสียเวลาในการคำนวณหางานที่เหมาะสม และไม่เหมาะกับระบบเวลาจริง

2.2.4 ความสำคัญของงาน (Priority)

ในการจัดลำดับการทำงาน งานต่าง ๆ มักจะถูกกำหนดค่าความสำคัญ เพื่อช่วยให้งานที่ต้องการความรวดเร็วของผลตอบมากกว่าได้ทำก่อน ส่วนงานที่มีค่าความสำคัญเท่ากันอาจใช้วิธีการจัดลำดับการทำงานแบบ Cyclic หรือ Round-robin อีกต่อหนึ่งก็ได้ ในการกำหนดค่าความสำคัญ อาจกำหนดแบบตายตัวโดยผู้ใช้ตั้งแต่แรก หรือมีการเปลี่ยนแปลงภายหลังก็ได้ สำหรับการเปลี่ยนแปลงค่าความสำคัญอาจทำได้โดยการคำนวณหรือโดยผู้ใช้เป็นต้น

2.2.5 Pre-emptive

มีลักษณะคือ งานที่มีความสำคัญมากกว่าจะสามารถขัดจังหวะการทำงานของงานที่มีความสำคัญน้อยกว่าได้ โดยสามารถทำได้ในหลายวิธี เช่น 1) เรียกตัวจัดการทุก ๆ คาบเวลาเล็ก ๆ เพื่อคอยตรวจสอบให้งานที่มีความสำคัญสูงสุดทำก่อนเสมอ หรือ 2) ใช้การตั้งเวลาเพื่อเรียกตัวจัดการ ในเวลาที่งานที่มีความสำคัญสูงที่สุดต้องการจะทำพอดี วิธีนี้จะทำให้งานที่มีความสำคัญสูงสุดเริ่มทำตรงตามเวลาที่ต้องการพอดี และไม่เสียเวลาในการเรียกตัวจัดการบ่อยครั้งจนเกินไป

2.2.6 Rate-monotonic

การจัดลำดับการทำงานแบบนี้เป็นการกำหนดค่าความสำคัญแปรผันตามความถี่ที่งานต้องการจะทำ เช่นงานที่ต้องการจะทำทุก ๆ 1 วินาที จะมีความสำคัญมากกว่างานที่ต้องการทำทุก ๆ 2 วินาที เป็นต้น

2.2.7 Earliest deadline

เป็นการจัดลำดับความสำคัญแบบไม่ตายตัว คือความสำคัญของงานขึ้นกับเวลาที่ต้องทำงานให้เสร็จ ถ้างานไหนต้องการทำให้เสร็จเร็วที่สุดก็จะมีความสำคัญมากที่สุด

2.2.8 Least slack

มีลักษณะใกล้เคียงกับแบบ Earliest deadline แต่ต่างกันตรงที่วิธีนี้จะนำเอาเวลาที่ต้องการใช้งานซีพียูมาคิดด้วย และค่าความสำคัญจะขึ้นอยู่กับ เวลาที่เหลือจากการหักลบเวลาที่ต้องการใช้งานซีพียูออกจากเวลาที่ต้องการทำงานให้เสร็จ ซึ่งจะได้จุดของเวลาที่น้อยที่สุดที่เมื่อเริ่มทำงานแล้วจะเสร็จทันเวลาพอดี ถ้าค่าเวลาที่ได้ยิ่งมีค่าน้อย งานนั้นก็จะยิ่งมีความสำคัญมากขึ้น

2.3 โครงสร้างความสำคัญ (Priority Structure)

โดยทั่วไปแล้ว งานต่าง ๆ จะถูกแบ่งออกเป็น 3 กลุ่ม ใหญ่ ๆ ตามความสำคัญหรือความเร็วในการสนองตอบที่ต้องการดังรูปที่ 2.1 คือ 1) ระดับอินเทอร์รัพ (Interrupt level) 2) ระดับสัญญาณนาฬิกา (Clock level) 3) ระดับฐาน (Base level) เพื่อเป็นประโยชน์ในการจัดลำดับการทำงานก่อนหลัง

2.3.1 ระดับอินเทอร์รัพ (Interrupt level)

งานในกลุ่มนี้เป็นงานที่ต้องการความเร็วในการสนองตอบเร็วที่สุด ในระดับของ มิลลิวินาที โดยทั่วไปการอินเทอร์รัพจะแบ่งเป็นหลายระดับย่อยตามความสำคัญ ระดับที่ความสำคัญสูงกว่าสามารถอินเทอร์รัพระดับที่ต่ำกว่าได้ และจะมีรูทีนมารองรับการอินเทอร์รัพแต่ละระดับต่าง ๆ กัน ซึ่งส่วนใหญ่จะเป็นการทำงานสั้น ๆ เท่าที่จำเป็น แล้วส่งผ่านงานส่วนที่เหลือให้ทำต่อในระดับนาฬิกาหรือระดับฐานต่อไป

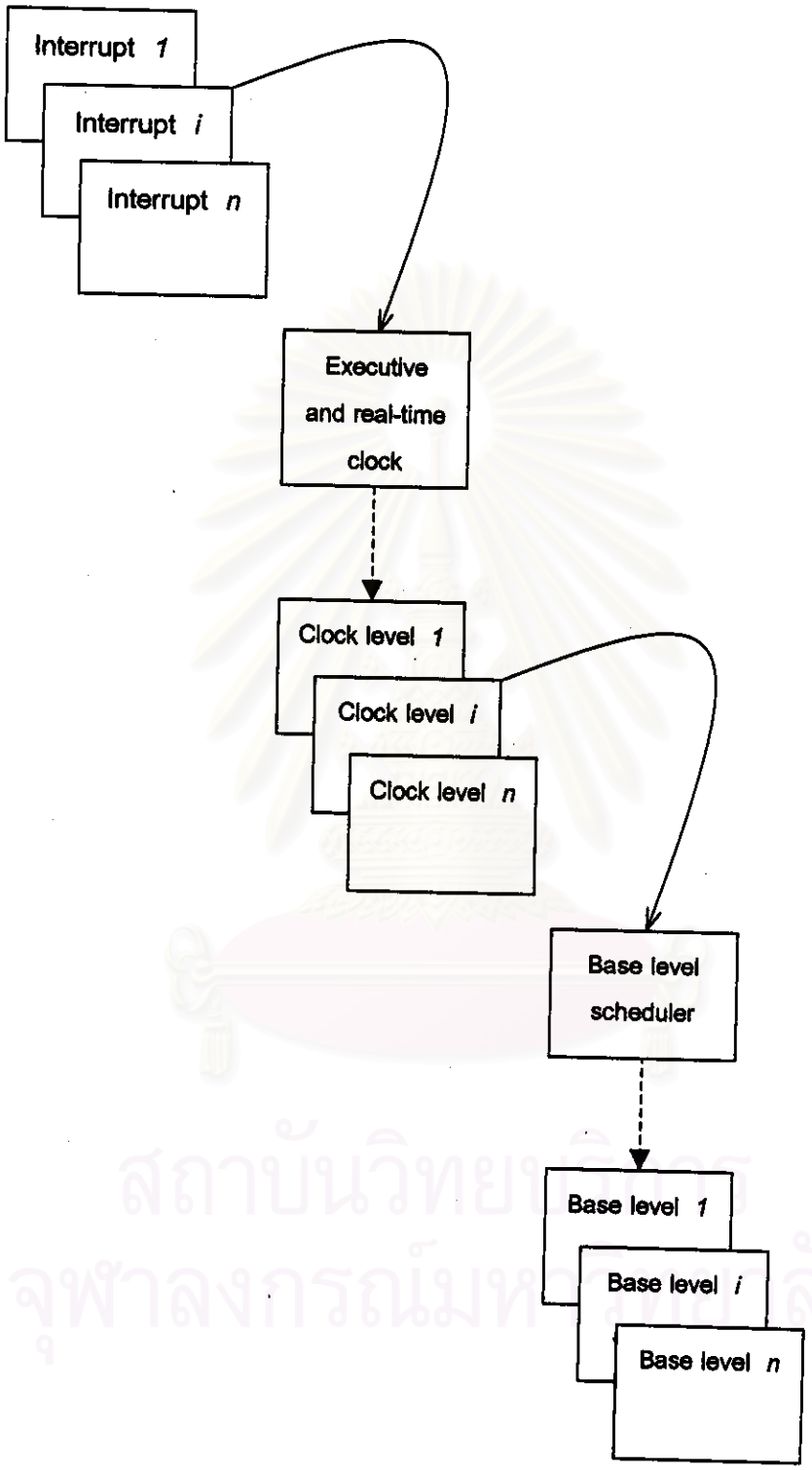
ในงานระดับอินเทอร์รัพ จะมีหนึ่งงานซึ่งรองรับการอินเทอร์รัพจากสัญญาณนาฬิกาของเครื่อง รูทีนนี้จะคอยแก้ไขเวลาของระบบปฏิบัติการและเรียกตัวจัดการงาน (Scheduler) ซึ่งจะทำหน้าที่ตัดสินใจว่าขณะนั้นควรจะทำงานใดต่อไป โดยงานในที่นี้จะหมายถึงงานในระดับนาฬิกาและระดับฐาน

2.3.2 ระดับนาฬิกา (Clock level)

งานในกลุ่มนี้เป็นงานที่ต้องการการทำงานแบบเวลาจริงหรือเป็นงานที่มีข้อจำกัดทางเวลา เช่นทำงานทุก ๆ 10 มิลลิวินาที เป็นต้น งานในกลุ่มนี้มักมีการจัดลำดับงานแบบ Cyclic, round-robin, pre-emptive, earliest deadline, least slack ตามความต้องการของงานแต่ละงานนั้น ๆ

2.3.3 ระดับฐาน (Base level)

งานในกลุ่มนี้เป็นงานที่ไม่มีข้อจำกัดทางเวลาหรืออาจเป็นงานที่มีข้อจำกัดทางเวลาแบบไม่เข้มงวด งานในกลุ่มนี้อาจมีการจัดลำดับแบบ Round-robin หรือ Nice level เพื่อเพิ่มประสิทธิภาพโดยรวมของระบบ



รูปที่ 2.1 ระดับความสำคัญในระบบปฏิบัติการแบบเวลาจริง

โดยทั่วไปในการทำงานแบบเวลาจริงมักแบ่งส่วนงานออกเป็นหลาย ๆ ระดับความสำคัญ โดยให้งานในระดับอินเทอร์รัพและระดับนาฬิกาจะมีขนาดเล็ก เช่นงานในการรับส่งข้อมูลจากอุปกรณ์ภายนอก งานการควบคุมเป็นต้น และงานในส่วนที่ไม่มีข้อจำกัดทางเวลา เช่นการแสดงผลข้อมูล การจัดเก็บข้อมูลลงไฟล์ การประมวลผลข้อมูลเพื่อวิเคราะห์ จะถูกกำหนดให้เป็นงานในระดับฐาน

2.4 การจัดการงาน (Task Management)

โดยปกติตัวจัดการงานจะมีหน้าที่หลัก ๆ 3 อย่างคือ 1) คอยเก็บสถานะของงานแต่ละงานว่าอยู่ในสถานะใด 2) คอยจัดลำดับการทำงานหรือจองการใช้งานซีพียูให้กับแต่ละงาน และ 3) สับเปลี่ยนการใช้งานซีพียูในแต่ละงาน ซึ่งจะต้องเก็บค่าสภาวะของงานขณะนั้น (เช่นค่า เรจิสเตอร์เป็นต้น) แล้วนำค่าสภาวะของงานใหม่นำมาใส่แทน

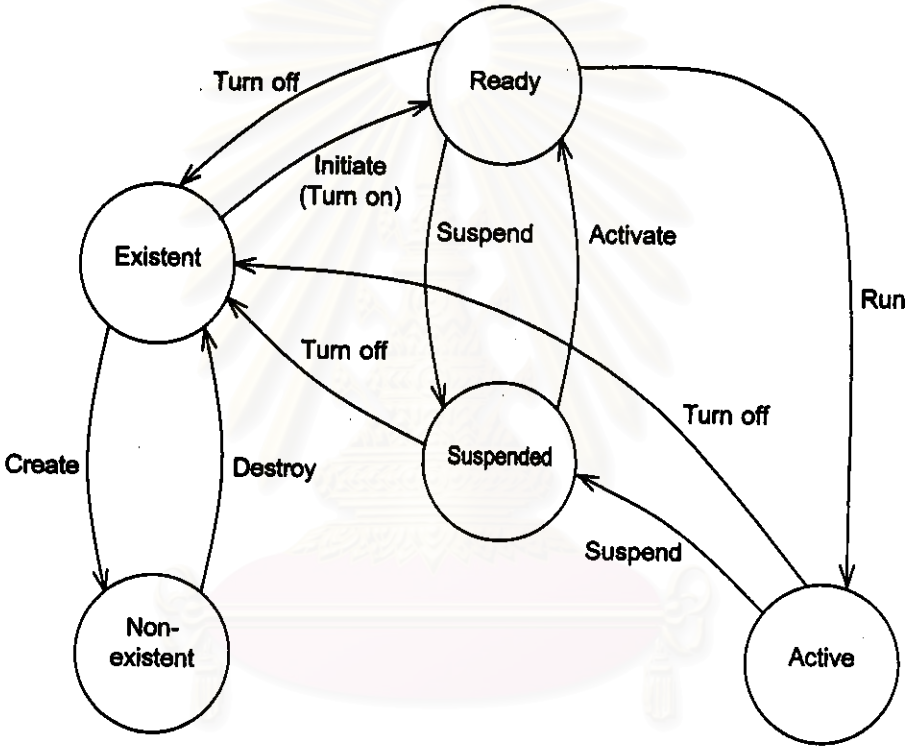
2.4.1 สภาวะของงาน (Task state)

ยกตัวอย่างในกรณีของเครื่องที่มีซีพียูเดียว ในเวลาหนึ่งจะมีเพียงหนึ่งงานเท่านั้นที่สามารถใช้งานซีพียูได้ สถานะของงานโดยส่วนใหญ่มักจะมีลักษณะดังในรูปที่ 2.2 โดยแต่ละสถานะสามารถอธิบายได้ดังนี้

- สถานะไม่มีตัวตน (non-existent, terminated) เป็นสถานะเริ่มต้นก่อนที่งานจะถูกกำหนดให้มีขึ้นในระบบ (create) หรือสถานะสุดท้ายที่งานถูกลบออกจากระบบ (destroy)
- สถานะมีตัวตน (existent, dormant, off) เป็นสถานะที่บ่งบอกว่ามีงานนั้นอยู่ในระบบ โดยประกอบด้วย 1) งานที่ถูกสร้างขึ้น (create) แต่ยังไม่ได้มีการจองทรัพยากรต่าง ๆ ที่ต้องการใช้ หรืออาจมีการจองทรัพยากรที่ต้องการ แต่ยังไม่ได้กำหนดค่าต่าง ๆ เช่นค่าความสำคัญ ความถี่ของงาน ให้สมบูรณ์ 2) งานที่ทำเสร็จแล้วหรือถูกสั่งหยุดทำงาน (turn off) แต่ยังไม่ได้คืนทรัพยากรต่าง ๆ ที่จองเอาไว้
- สถานะพร้อมทำงาน (ready, runnable, on) สถานะนี้ประกอบด้วย 1) งานใหม่ที่มีการจองทรัพยากรต่าง ๆ เช่นหน่วยความจำ รวมทั้งมีการกำหนดค่าความสำคัญ ความถี่ในการทำงาน หรือ เส้นตายในการทำงาน เรียบร้อยแล้ว (Initiate) และพร้อมที่จะทำงานได้ 2) งานที่พร้อมที่เปลี่ยนสถานะจากสถานะถูกยับยั้งไปสู่สถานะทำงาน เนื่องจากเงื่อนไขที่ยับยั้งการทำงานหมดไป
- สถานะถูกยับยั้ง (suspended, waiting) งานที่อยู่ในสถานะนี้เกิดจากงานที่อยู่ในสถานะทำงานหรือสถานะพร้อมทำงาน ถูกยับยั้งไว้ไม่ให้ทำงานเนื่องจากสาเหตุหลาย

ประการ เช่น รอการตอบรับจากอุปกรณ์ภายนอก รอการหน่วงเวลา หรืออาจรอให้ถึงคาบเวลาที่กำหนดค่อยทำงานต่อ เป็นต้น

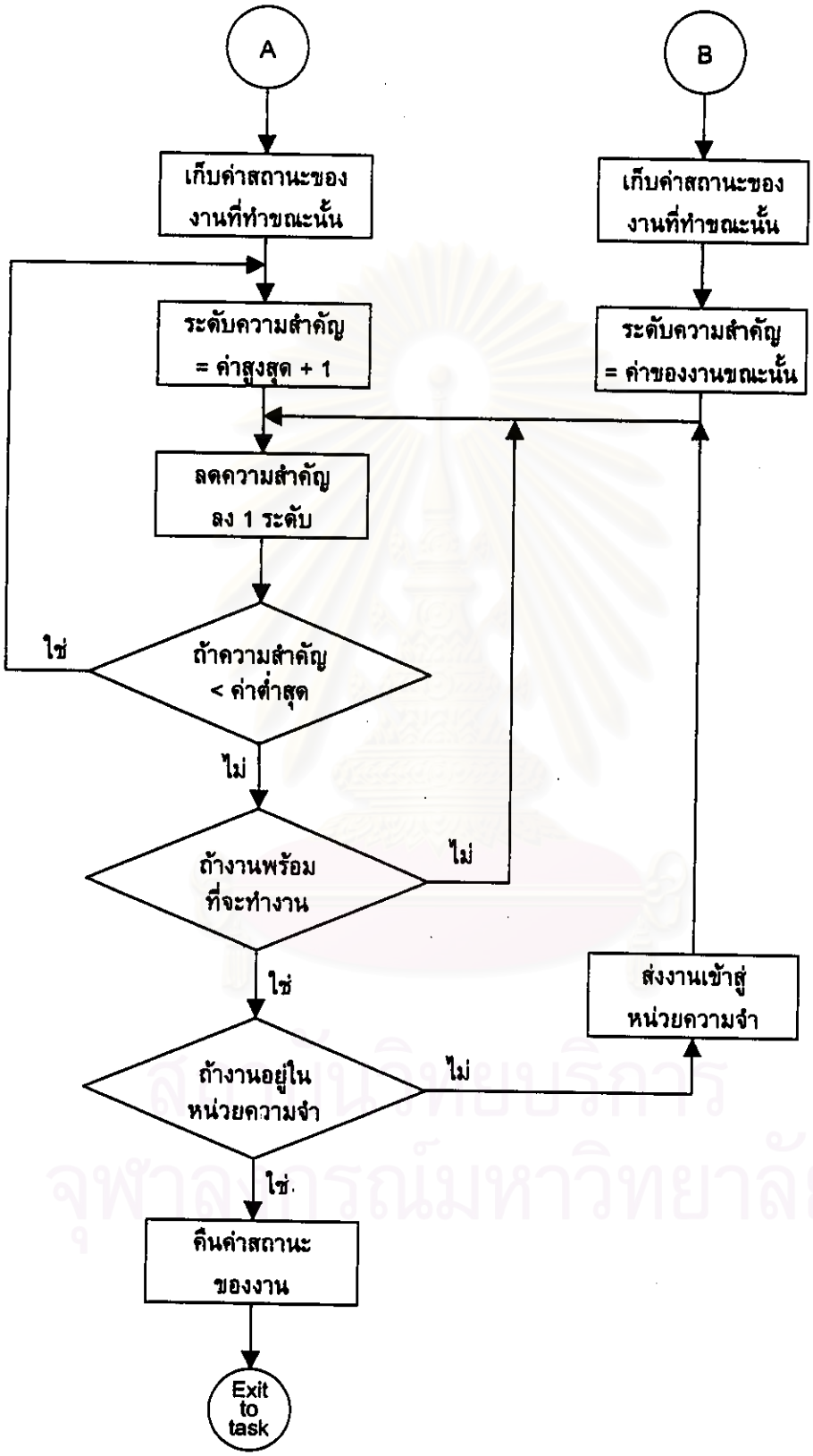
- สถานะทำงาน (active, running) เป็นสถานะที่ได้ใช้งานซีพียู โดยเป็นงานที่มีความสำคัญที่สุดหรือเป็นงานที่เหมาะสมที่สุดซึ่งตัวจัดการงาน (scheduler) เลือกมาจากงานในสถานะพร้อมทำงานขณะนั้น



รูปที่ 2.2 ตัวอย่างแผนภูมิสถานะของงาน

2.4.2 ตัวจัดการงาน (Scheduler)

โดยทั่วไปตัวจัดการงานจะถูกเรียกใช้ใน 2 เหตุการณ์ คือ 1) การอินเทอร์รัพของสัญญาณนาฬิกาและเมื่อจบรูทีนที่รองรับอินเทอร์รัพ และ 2) การถูกยับยั้งของงานด้วยสาเหตุต่าง ๆ เช่น การหน่วงเวลา การรอสัญญาณจากภายนอก เป็นต้น ผังการทำงานของตัวจัดการงานดังกล่าวสามารถแสดงได้ดังรูปที่ 2.3



รูปที่ 2.3 แสดงการทำงานของตัวจัดการงาน

ผังการทำงานของตัวจัดการงานดังกล่าวแสดงได้ดังรูปที่ 2.3 คือ ในกรณี A ตัวจัดการงานถูกเรียกจากอินเทอร์รัพของสัญญาณนาฬิกา และจะเริ่มต้นจากค่าความสำคัญสูงสุดก่อน แล้วตรวจสอบแต่ละงานในแต่ละระดับความสำคัญ นั่นคืองานที่มีความสำคัญสูงสุดจะถูกทำก่อนเสมอ ส่วนในกรณี B ตัวจัดการงานถูกเรียกจากการที่งานที่ทำในขณะนั้นถูกยับยั้งเนื่องจากการหน่วงเวลา หรือการรอการตอบรับจากอุปกรณ์ภายนอก กรณีนี้ จะเริ่มต้นจากค่าความสำคัญรองจากงานปัจจุบันหนึ่งค่า เพราะงานที่มีความสำคัญมากกว่าจะสามารถขัดจังหวะการทำงานของงานที่มีความสำคัญน้อยกว่าได้เสมอ

2.5 การจัดการหน่วยความจำ (Memory Management)

การจัดการหน่วยความจำแบ่งได้ออกเป็นแบบคงที่ (static allocation) และแบบเปลี่ยนแปลงได้ (dynamic allocation) โดยทั่วไปในการทำงานแบบเวลาจริงมักจะกำหนดในลักษณะคงที่มากกว่าในแบบเปลี่ยนแปลงได้ เนื่องจากการกำหนดแบบคงที่จะประหยัดเวลาในการสับเปลี่ยนงานมากกว่า มีความซับซ้อนและความผิดพลาดน้อยกว่า รวมทั้งในปัจจุบันหน่วยความจำก็มีราคาลดลงอย่างมาก แต่อย่างไรก็ตามในระบบที่ใช้งานแบบทั่วไปไม่เฉพาะเจาะจงมักเลือกใช้งานในแบบเปลี่ยนแปลงได้เพราะจะประหยัดทรัพยากรและมีความยืดหยุ่นมากกว่า

2.6 การตรวจสอบซึ่งกันและกัน (Mutual Exclusion)

ในระบบปฏิบัติการแบบหลายภาระกิจยอมให้มีการใช้งานทรัพยากรต่าง ๆ ร่วมกัน แต่ทรัพยากรบางอย่างไม่สามารถถูกใช้โดยงานในสถานะทำงานพร้อมกันได้ เช่นการใช้งานอุปกรณ์ที่ทำหน้าที่รับ-ส่งข้อมูลระหว่างคอมพิวเตอร์กับอุปกรณ์ภายนอก การใช้งานเครื่องพิมพ์ การใช้งานข้อมูลหรือตัวแปรร่วมกันเป็นต้น การใช้งานอุปกรณ์หรือตัวแปรที่มีลักษณะดังกล่าวจะต้องมีวิธีการในการที่จะตรวจสอบสถานะของทรัพยากรว่าพร้อมสำหรับใช้งานในขณะนั้นหรือไม่ ถ้าไม่พร้อมก็จะมีคิวสำหรับงานที่รอทรัพยากรนั้น ๆ และถ้าพร้อมก็จะต้องมีวิธีการในการกันไม่ให้งานอื่นมาใช้งาน วิธีการที่ใช้กันทั่วไปมีหลายแบบเช่น เซมาฟอร์ มอนิเตอร์ การตรวจสอบหัวท้าย [5] เป็นต้น

2.6.1 เซมาฟอร์ (Semaphore)

วิธีการนี้เป็นวิธีการที่ง่ายและใช้กันอย่างแพร่หลาย มีหลักการง่าย ๆ คือ ใช้แฟลกแทนสถานะของทรัพยากร เช่นในกรณีที่ทรัพยากรอยู่ในสถานะพร้อมใช้งาน แฟลกจะมีค่าเป็น 1 และเมื่อมีการใช้งานแฟลกจะถูกเปลี่ยนค่าให้เป็น 0 สำหรับในกรณีที่มีการสร้างคิวสำหรับรอ

การใช้งาน คิวจะเก็บลิสต์ของงานที่ต้องการใช้ทรัพยากร และค่าแฟลกอาจถูกลดลงตามจำนวนงานที่รออยู่ก็ได้ โดยเมื่อทรัพยากรนั้นพร้อมใช้งาน ตัวจัดการงานก็จะทำหน้าที่สับเปลี่ยนการทำงานไปยังงานที่รออยู่เหล่านั้น การที่จะตัดสินใจทำงานใดก่อนหรือหลังนั้นขึ้นอยู่กับวิธีการจัดลำดับงานที่เลือกใช้ แต่เซมาฟอร์ก็มีข้อเสียคือ ในการใช้งานจำเป็นจะต้องรู้รายละเอียดซึ่งมีความแตกต่างกันในแต่ละทรัพยากร และจะกระจายอยู่ตามโปรแกรมต่าง ๆ ที่ต้องการใช้งาน ทำให้เป็นภาระในการพัฒนาโปรแกรมและการแก้ไขโปรแกรมในภายหลัง โดยเฉพาะอย่างยิ่งในการพัฒนาระบบใหญ่ ๆ

2.6.2 มอนิเตอร์ (monitor)

เนื่องมาจากข้อเสียในกรณีของการใช้งานเซมาฟอร์ จึงได้มีผู้คิดวิธีในการป้องกันการใช้งานทรัพยากรซ้ำซ้อนในอีกแนวทางหนึ่ง โดยมีหลักการคือ มีมอนิเตอร์เป็นเซตของฟังก์ชันที่ใช้ในการเข้าถึงข้อมูลหรือทรัพยากรนั้น ๆ โดยฟังก์ชันเหล่านี้จะถูกรวมอยู่ภายในมอดูลซึ่งทำหน้าที่จัดการการใช้งานทรัพยากรแทนงานอื่น ตัวอย่างมอนิเตอร์สำหรับข้อมูลที่ใช้ร่วมกันอย่างง่ายสามารถออกแบบได้ดังนี้ คือ ประกอบด้วยฟังก์ชันสำหรับอ่านข้อมูล และฟังก์ชันสำหรับเขียนข้อมูล ในกรณีที่ต้องการเขียนข้อมูลโดยที่มีงานอื่นกำลังอ่านหรือเขียนอยู่ มอนิเตอร์จะหยุดรอจนกว่าข้อมูลจะพร้อม และในกรณีที่มีความต้องการในการเขียนข้อมูลเข้ามาพร้อมกันมาก ๆ มอนิเตอร์จะทำหน้าที่ในการจัดลำดับการทำงานว่างานใดควรทำก่อนหรือหลัง

2.6.3 การตรวจสอบหัวท้าย

วิธีการนี้เป็นวิธีการสำหรับใช้กับการเข้าถึงข้อมูลในหน่วยความจำร่วมกันเท่านั้น โดยสำหรับในกรณีของการใช้ข้อมูลร่วมกัน ข้อมูลจะถูกกำหนดให้มีตัวแปรตัวหนึ่งเป็นหัว head และอีกตัวแปรหนึ่งเป็นท้าย tail เริ่มแรกทั้งสองตัวแปรจะถูกตั้งให้มีค่าเท่ากันเช่น head = 0, tail = 0 เมื่อมีการเข้าถึงข้อมูลโดยงานหรือโปรแกรมใด ๆ ก็ตาม ตอนแรกตัวแปรหัวจะถูกเพิ่มค่าขึ้นอีกหนึ่ง head = head + 1 ก่อนที่จะทำการแก้ไขตัวข้อมูล และหลังจากทำการแก้ไขข้อมูลแล้วก็จะตั้งค่าของตัวแปรหัวและท้ายให้เท่ากัน tail = head ในระหว่างที่มีการแก้ไขข้อมูลนี้ ถ้ามีงานอื่นต้องการเข้าถึงข้อมูล จะต้องตรวจสอบค่าของตัวแปรหัวและท้ายว่าเท่ากันหรือไม่ ถ้าไม่เท่ากันจะต้องรอจนกว่าตัวแปรทั้งสองจะเท่ากันซึ่งเป็นข้อเสียของวิธีนี้ เพราะเวลาที่ต้องรอจะไม่แน่นอน (ถ้าไม่รอแบบ busy-wait เวลาที่รออย่างน้อยจะเท่ากับหนึ่งคาบเวลา) ข้อเสียอีกอย่างหนึ่งคือความไม่รัดกุม เนื่องจากขณะทำการเพิ่มค่าของตัวแปร head ซึ่งไม่สามารถทำได้ภายในคำสั่งของเครื่อง (machine instruction) คำสั่งเดียวอาจถูกอินเทอร์รัพจากสัญญาณนาฬิกาและสับเปลี่ยนไปทำงานอื่นได้ ดังนั้นจะต้องสั่งห้ามการอินเทอร์รัพระหว่างการเพิ่มค่าตัวแปร head เพื่อไม่ให้เกิดปัญหานี้ขึ้น อย่างไรก็ตามในงานที่มีความถี่ไม่สูงนักและความผิดผลาดสามารถยอมรับได้ มักจะเลือกใช้วิธีนี้เพราะเป็นวิธีที่ง่ายและไม่สั่งห้ามการอินเทอร์รัพ

2.7 การติดต่อระหว่างงาน (Inter-task Communication)

ในการทำงานในระบบหลายภาระกิจ (multi-tasking) จำเป็นจะต้องมีการติดต่อสื่อสารกันระหว่างงานแต่ละงานที่เกี่ยวข้องกัน แบ่งได้เป็น 2 ประเภทคือ การสื่อสารเพื่อให้เกิดความสอดคล้องกัน (Synchronization) และการสื่อสารเพื่อส่งผ่านข้อมูล (data transfer)

2.7.1 การสื่อสารเพื่อให้เกิดความสอดคล้องกัน (Synchronization)

วิธีการสื่อสารเพื่อให้เกิดความสอดคล้องกันที่แพร่หลายมากที่สุดคือ การใช้สัญญาณ (signal) เช่นมีงาน ก. และ งาน ข. ทำงานพร้อมกันและมีบางส่วนของงาน ข. ต้องการใช้ข้อมูลที่ได้จากงาน ก. นั่นคืองาน ข. จะรอรับสัญญาณจากงาน ก. เมื่องาน ก. ทำงานเสร็จก็จะส่งสัญญาณให้กับงาน ข. เพื่อทำงานต่อไปเป็นต้น ตัวอย่างของการสร้างสัญญาณแบบง่าย ๆ จะคล้ายคลึงกับเซมาฟอร์คือ ใช้แฟลกที่มีค่า 0 กับ 1 ในกรณีที่ไม่มีสัญญาณให้แฟลกมีค่าเป็น 0 การส่งสัญญาณจะหมายถึงการเปลี่ยนค่าแฟลกให้เป็น 1 และการรอรับสัญญาณก็จะหมายถึงการรอให้ค่าแฟลกเป็น 1 นั่นเอง

2.7.2 การสื่อสารเพื่อส่งผ่านข้อมูล (data transfer)

นอกจากความสอดคล้องในการทำงานระหว่างกันแล้ว ยังมีความต้องการในการที่จะแลกเปลี่ยนข้อมูลระหว่างกันอีกด้วย ตัวอย่างของการแลกเปลี่ยนข้อมูลระหว่างกันมีหลายวิธีเช่น การใช้ตัวแปรร่วมกัน การใช้หน่วยความจำร่วมกัน โดยแต่ละงานที่ต้องการอ่านหรือแก้ไขข้อมูลจะทำที่ที่เดียวกัน วิธีนี้จะต้องนำการตรวจเช็คซึ่งกันและกัน (mutual exclusion) มาใช้เพื่อป้องกันความผิดผลาดของข้อมูล ส่วนอีกวิธีหนึ่งเป็นการส่งข้อมูลแบบไม่แก้ไขข้อมูลเก่าโดยตรง ลักษณะของการส่งข้อมูลแบบนี้จะเป็นแบบหนึ่งต่อหนึ่ง ตัวอย่างเช่นการใช้คิวหรือลิงลิสต์ ข้อมูลที่ถูกส่งจะเข้าไปอยู่ในคิวและยังคงอยู่จนกว่าจะถูกอ่านออกไป ข้อมูลใหม่ที่ถูกส่งเข้ามาจะอยู่ต่อท้ายข้อมูลเก่า และถูกอ่านออกไปในลักษณะเข้าก่อนออกก่อน (First-In-First-Out)

บทนี้ได้อธิบายเกี่ยวกับสิ่งที่จะต้องคำนึงถึงสำหรับระบบปฏิบัติการแบบเวลาจริง เพื่อจะใช้ในการพิจารณาและเลือกระบบปฏิบัติการที่จะนำมาใช้สำหรับงานวิทยานิพนธ์นั่นคือ RT-Linux ซึ่งเป็นระบบปฏิบัติการที่ดัดแปลงจาก Linux โดยเพิ่มเฉพาะส่วนของความสามารถด้านการทำงานเวลาจริงแบบเข้มงวด (hard real-time) เข้าไป ส่วนความสามารถด้านอื่น ๆ ที่มีอยู่เดิมของ Linux ยังคงสามารถทำงานได้ต่อไป สำหรับการพิจารณาดังกล่าวจะอยู่ในบทต่อไปโดยจะกล่าวถึงความสามารถในการทำงานแบบเวลาจริงของทั้ง Linux ปกติและ Linux ที่มีการดัดแปลงแล้วหรือ RT-Linux