


Embedded Systems Programming on STM32 MCU

การโปรแกรมระบบสมองกลฝังตัวบน

ไมโครคอนโทรลเลอร์ STM32


ครั้งที่ 6 : ปัญหาภายใต้การทำงานแบบหลายภารกิจ


 แนวทางการจัดการการขัดจังหวะบนระบบคอมพิวเตอร์โดยทั่วไป


 Preemptive vs. Non-preemptive multitasking


-  ความแตกต่างระหว่างการจัดการแบบ preemptive และ non-preemptive multitasking

-  การจัดการ preemptive ใน FreeRTOS ของ STM32

-  การใช้ osThreadYield() เพื่อช่วยการทำงานของ FreeRTOS

 สภาวะแข่งขัน และส่วนวิกฤติ

-  ความหมายของสภาวะแข่งขัน

-  ส่วนวิกฤติ

-  การใช้งาน mutex lock เบื้องต้น



Asst.Prof. Thanwa SRIPRAMONG

PRESENTER

TODAY TOPIC IS

Multitasking Issues

การจัดการการขัดจังหวะ (interrupt) บน MCU

🌿 โดยปกติแล้ว ระบบปฏิบัติการแบบหลายภารกิจ หลายผู้ใช้ (multitasking / multiuser) จะไม่อนุญาตให้โปรเซสผู้ใช้เข้าถึงอุปกรณ์ทางฮาร์ดแวร์โดยตรง แต่จัดการโดยระบบปฏิบัติการ

💡 การจัดการฮาร์ดแวร์ จะกระทำผ่าน system call (ฟังก์ชันบริการของระบบปฏิบัติการ)

🌈 เท่ากับว่า การจัดการ interrupt โดยทั่วไปจะไม่สามารถเข้าถึงได้โดยโปรเซสผู้ใช้

🌈 เพื่อความปลอดภัยของการทำงานในองค์กรรวม (โปรเซสที่ทำงานผิดพลาด หรือโปรเซสประสงค์ร้าย จะไม่ส่งผลกระทบต่อโปรเซสอื่น)

🌈 ระบบปฏิบัติการใช้สัญญาณ (signal) หรือเหตุการณ์ (event) ในการส่งการจากผู้ใช้ แทนการเข้าถึง interrupt โดยตรง



Asst.Prof. Thanwa SRIPRAMONG
PRESENTER

TODAY TOPIC IS

Multitasking Issues

การจัดการการขัดจังหวะ (interrupt) บน MCU

🌱 MCU ที่มีทรัพยากรน้อย และมีโปรเซสผู้ใช้หลักเพียงตัวเดียว (อาจรองรับการทำงานแบบ multitasking ในรูปของ multithreading – แบ่งการทำงานของโปรเซสเป็นหลายเธรด) มักจะอนุญาตให้โปรเซสเข้าถึงกลไกทางฮาร์ดแวร์ได้เต็มที่

💡 เนื่องจากมีเพียงโปรเซสเดียวที่ทำงานอยู่ การออกแบบการทำงานของเธรดให้ทำงานอย่างถูกต้อง ตกอยู่กับผู้พัฒนา

💡 ผู้ผลิต MCU อาจออกแบบไลบรารีช่วยในการเข้าถึงฮาร์ดแวร์ส่วนต่างๆ ให้กับผู้ใช้ (ผู้ใช้อาจเข้าถึงฮาร์ดแวร์โดยตรง หรือผ่านไลบรารีก็ได้)

💡 ระบบปฏิบัติการ มีหน้าที่สนับสนุนการทำงานอื่นๆ เช่น การจัดการโปรเซสและเธรด

🌈 กรณีของ FreeRTOS ทำหน้าที่เป็นไลบรารีเสริมด้านการจัดการโปรเซสและการจัดการหน่วยความจำ ให้กับโปรเซสผู้ใช้

🌱 การพัฒนาซอฟต์แวร์บน MCU โดยทั่วไป ยอมให้ผู้พัฒนาใช้กลไกการขัดจังหวะได้ด้วยตนเอง

💡 MCU แม้กระทั่งเช่นที่ใช้ใน Raspberry Pi อนุญาตให้ผู้พัฒนาใช้กลไก interrupt กับ GPIO และ I/O ต่างๆ ได้เช่นกัน

💡 เป็นภาระของผู้พัฒนาที่ต้องระมัดระวังการใช้งาน interrupt ด้วยตนเอง



Asst.Prof. Thanwa SRIPRAMONG

PRESENTER

TODAY TOPIC IS

Multitasking Issues

ข้อระมัดระวังในการจัดการการขัดจังหวะ บน MCU

🌿 ควบคุมเวลาการทำงานภายในฟังก์ชันบริการการขัดจังหวะ ให้ใช้เวลาน้อยที่สุด

💡 เวลาที่เสียไปในการทำงานของ ISR เท่ากับเวลาที่เสียไปที่โปรเซสหลักไม่ได้ทำงาน

🌈 การทำงานตามปกติจะช้าลงถ้ามีการขัดจังหวะเกิดขึ้นบ่อยมาก

💡 ลดโอกาสการเกิด reentrant

🌈 สถานะที่เกิดการขัดจังหวะเดิมซ้ำซ้อนในขณะที่ทำงานอยู่ใน ISR อาจทำให้จัดการข้อมูลผิดพลาด (เช่น กำลังอ่านข้อมูลตัวหนึ่งจากพอร์ตอนุกรม แต่ยังไม่ทันนำค่าไปจัดเก็บ ต้องอ่านค่าใหม่ซ้ำเข้ามา และอาจทำให้ค่าเก่าสูญหาย/เขียนทับ หรืออื่นๆ เป็นต้น)

🌈 อาจลดปัญหาด้วยการยกเลิกการขัดจังหวะชั่วคราว แต่ก็ไม่ได้แก้ปัญหาเรื่องการประมวลผลข้อมูล I/O ไม่ทัน

🌿 ส่วนการจัดการที่ไม่ต้องเป็น real-time อาจเลี่ยงไปใช้การจัดการรูปแบบอื่นแทน

💡 อาจใช้การหยั่งสัญญาณ (polling) แทน

💡 หรือผสมผสานกับการสร้างเธรดที่มีลำดับความสำคัญต่ำ

🌿 หรืออาจหันไปใช้วิธีการอื่น เช่น DMA ถ้าฮาร์ดแวร์รองรับ



Asst.Prof. Thanwa SRIPRAMONG
PRESENTER

TODAY TOPIC IS

Multitasking Issues

Preemptive vs. Non-preemptive multitasking

- 🌿 วัฏจักรการทำงานของโปรเซส / เธรด อยู่ในรูปของการประมวลคำสั่ง (CPU burst cycle) สลับกับการรับเข้า/ส่งออก ข้อมูล (I/O burst cycle)
 - 💡 การประมวลคำสั่ง อาศัยการทำงานโดยหน่วยประมวลผลกลางทำหน้าที่ประมวลชุดคำสั่งต่างๆ
 - 💡 การรับเข้า/ ส่งออก ข้อมูลกับอุปกรณ์รอบข้าง ซึ่งทำงานช้า หน่วยประมวลผลกลางจะต้องรอเวลา I/O ทำงานจนเสร็จจึงได้ข้อมูลไปดำเนินการต่อ
 - 🌈 ในทางปฏิบัติ อาจผลักระบบการจัดการให้กับ DMA เพื่อรับ/ส่ง ข้อมูลกับหน่วยความจำหลัก
 - 🌈 หรือพักเธรดปัจจุบันไว้ก่อน รอให้ I/O ทำงานเสร็จ เพื่อเอาเวลาซีพียูไปประมวลเธรดอื่นแทน
- 🌿 ระบบปฏิบัติการ จึงอาศัยจังหวะต่างๆ เพื่อใช้ในการสลับงาน (โปรเซส/เธรด) ไปมา
 - 💡 เมื่อโปรเซส/เธรด ร้องขอบริการของระบบปฏิบัติการ (system call) เพื่อติดต่อ I/O หรือบริการอื่นๆ (ยกตัวอย่างเช่น ฟังก์ชันรอเวลา delay())
 - 💡 เมื่อมีการทำงานตามฟังก์ชันบริการการขัดจังหวะ (ISR) ที่ระบบปฏิบัติการสร้างเตรียมไว้ให้เพื่อจัดการกับ I/O ระบบปฏิบัติการอาจสลับงานในจังหวะนี้ได้
 - 💡 เมื่อเกิดการขัดจังหวะ(interrupt) ในระบบ (เช่นการขัดจังหวะจากฐานเวลาระบบ)





Asst.Prof. Thanwa SRIPRAMONG
PRESENTER

TODAY TOPIC IS



Multitasking Issues

Preemptive vs. Non-preemptive multitasking

Non-preemptive multitasking

-  การสลับงานเกิดขึ้นก็ต่อเมื่อทาส์กจบรอบการประมวลผล (CPU burst cycle) และต้องการบริการจากระบบปฏิบัติการ (system call) เท่านั้น
-  ถ้าทาส์กเกิดการทำงานผิดพลาด (เช่นวนรอบ while() อนันต์) อาจทำให้ระบบทั้งหมดหยุดทำงานได้

Preemptive multitasking

-  ระบบปฏิบัติการใช้กลไกการขัดจังหวะ เพื่อเข้ามาสลับงานแม้ว่าทาส์กปัจจุบันยังคำนวณไม่เสร็จ
-  ถ้าทาส์กบางตัวไม่ยอมจบ CPU burst cycle ปัจจุบัน ระบบปฏิบัติการสามารถบังคับให้หยุดคำนวณชั่วคราว เพื่อให้ทาส์กอื่นทำงานได้

หมายเหตุ งานหรือทาส์ก (task) คือการดำเนินงานองค์หนึ่งๆ ที่ถูกจัดการโดยระบบปฏิบัติการ ในทางปฏิบัติสำหรับระบบปฏิบัติการที่ไม่รองรับเธรดจะหมายถึงโปรเซส(process) ส่วนระบบปฏิบัติการที่รองรับเธรดจะมองหน่วยการทำงานเป็นเธรด (thread)



Asst.Prof. Thanwa SRIPRAMONG
PRESENTER

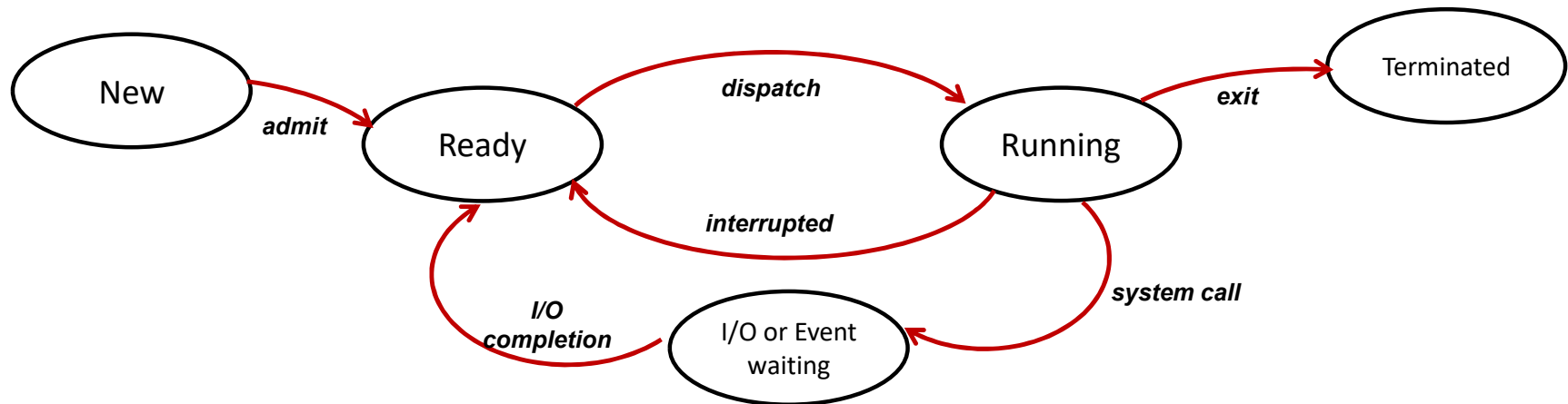
TODAY TOPIC IS

Multitasking Issues

Multitasking – switching context

🌱 จังหวะระหว่างการทำงานของโปรเซส/เธรด ที่เอื้อให้เกิดการสลับทาส์ก

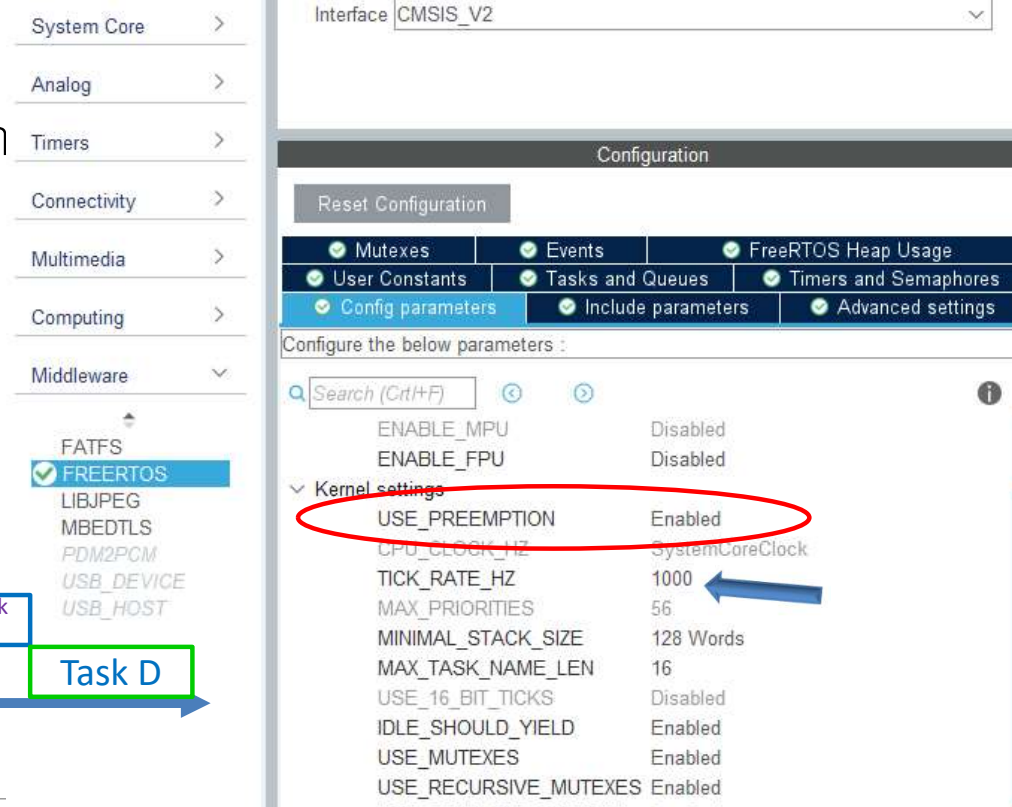
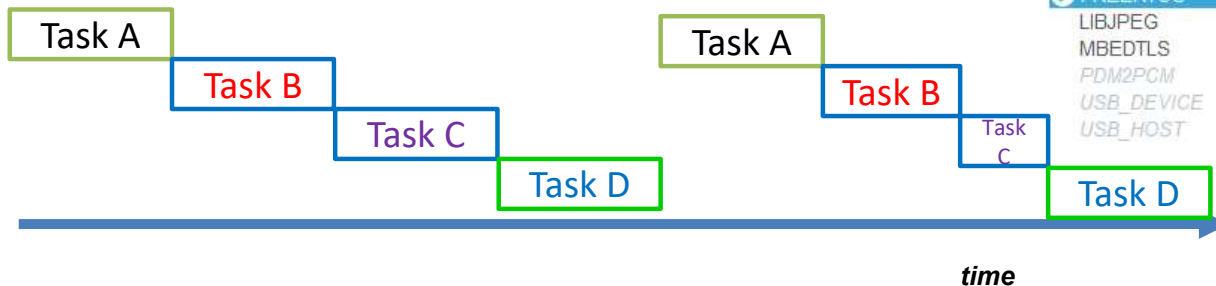
- 🔵 โปรเซสร้องขอ system call (non-preemptive)
- 🔵 เกิด I/O interrupt เมื่อ I/O ทำงานเสร็จ (เพื่อส่งผลลัพธ์ให้โปรเซสที่รอคอย) (preemptive)
- 🔵 โปรเซสจบการทำงาน (ร้องขอ exit()) (non-preemptive)
- 🔵 โปรเซสถูกขัดจังหวะจากกลไกอื่นใด (เช่น ฐานเวลาระบบ) (preemptive)



Preemptive multitasking in FreeRTOS

FreeRTOS ใช้การสลับทาสก์ในรูปของ Round-Robin (RR)

- เรียกอีกอย่างว่า time-slicing
- ใช้ SysTick (timer หลักของ Arm) เป็นฐานเวลา
- รองรับการสลับทาสก์แบบ preemptive




Asst.Prof. Thanwa SRIPRAMONG
PRESENTER


TODAY TOPIC IS


Multitasking Issues


Preemptive multitasking in FreeRTOS

Round-Robin (RR)

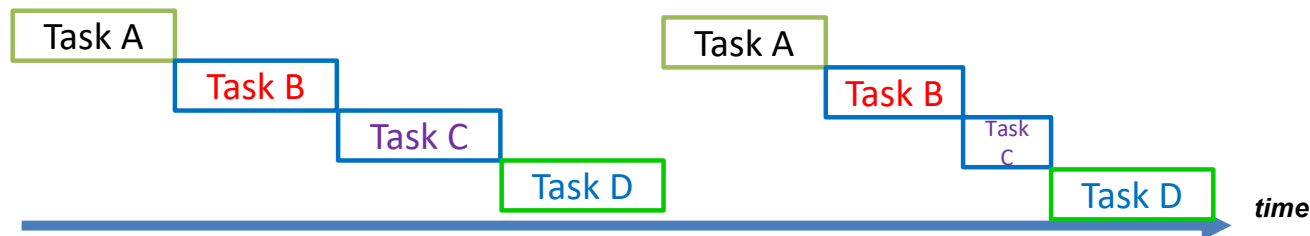
 ระบบปฏิบัติการจะเข้ามาสลับงานเป็นระยะ (ผ่านการขัดจังหวะของ SysTick)

 แต่ละทาสก์มีค่าลำดับความสำคัญที่อาจแตกต่างกันไปได้

 ถ้าสองทาสก์มีค่าลำดับความสำคัญที่เท่าเทียมกัน จะได้สัดส่วนเวลาในการทำงานเท่าๆ กัน

 ทาสก์ที่มีลำดับความสำคัญมากกว่า จะสามารถแข่งคิวประมวลผลได้ (หากรอพร้อมกับทาสก์ที่มีลำดับความสำคัญต่ำกว่า)

Task Name	Priority	Stack...	Entry Funct...	Code Gene...	Parameter	Allocation	Buffer Name	Control Blo.
runningLED	osPriorityNormal	128	runningLED...	Default	NULL	Dynamic	NULL	NULL
ultrasonic	osPriorityLow	128	ultrasonicT...	Default	NULL	Dynamic	NULL	NULL



Asst.Prof. Thanwa SRIPRAMONG
PRESENTER

TODAY TOPIC IS

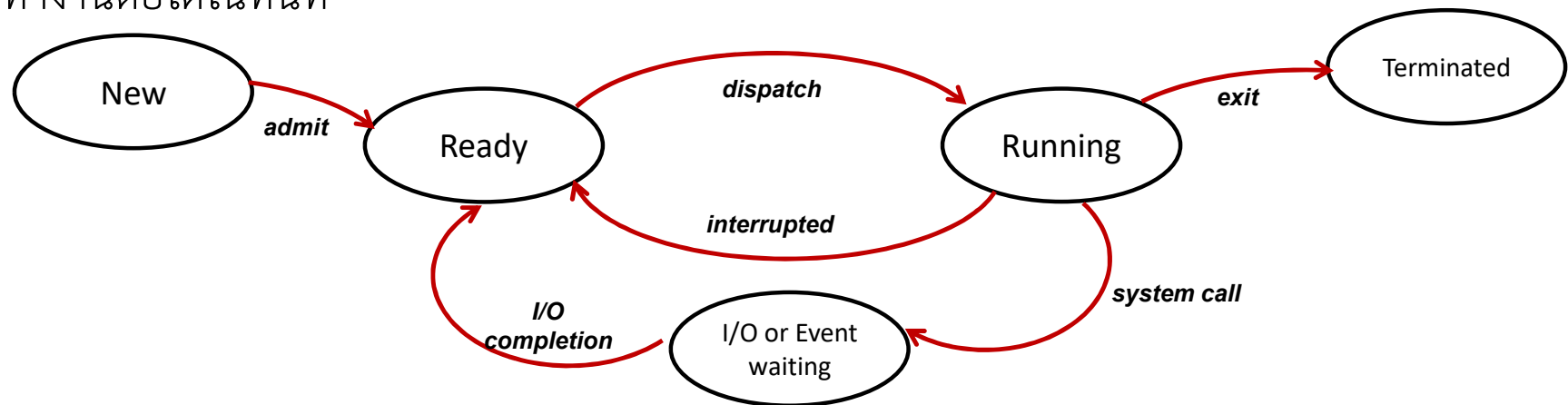
Multitasking Issues

Preemptive multitasking in FreeRTOS

Round-Robin (RR)

🔵 ในกรณีที่ทาสก์ที่มีลำดับความสำคัญต่ำกว่ากำลังประมวลผลอยู่ แต่มีทาสก์ที่มีลำดับความสำคัญสูงกว่ามารอคิว จะสลับเอาทาสก์ลำดับความสำคัญต่ำกว่าออกเพื่อให้ทาสก์ที่มีลำดับความสำคัญสูงกว่าประมวลผลก่อน

🔵 ทาสก์อาจใช้เวลาคำนวณ (CPU burst) น้อยกว่า time slot ของ RR ได้ ในกรณีเช่นนี้ ทาสก์อื่นที่รออยู่จะทำงานต่อได้ในทันที



คำแนะนำในการใช้ FreeRTOS

🌿 กำหนดให้แต่ละเธรด (ทาสก์) มีลำดับความสำคัญเท่าที่จำเป็น

💡 เธรดที่ไม่ต้องการความเป็น "ทันเวลา" (real-time) สูง ให้กำหนดลำดับความสำคัญต่ำ

🎨 รวมถึงเธรดที่อาจต้องเสียเวลาคำนวณนาน (มี CPU burst time ที่สูง) เพื่อเปิดโอกาสให้เธรดอื่นได้มีโอกาสทำงานบ้าง

🌿 หลีกเลี่ยงการคำนวณที่ไม่จำเป็น

💡 ตัวอย่างเช่น การใช้ while(1) วนรอบอ่านค่าจนกว่าจะได้สถานะที่ต้องการ

🎨 ไม่สร้างสถานการณ์ที่ทำให้มีเธรดที่มี CPU Burst time นานๆ โดยไม่จำเป็น

🎨 หันไปใช้กลไกอื่นแทน เช่นการขัดจังหวะจาก I/O เมื่อทำงานได้ข้อมูลเรียบร้อยแล้ว

🌿 ลด CPU burst time ลงถ้าทำได้ โดยการเรียกใช้บริการของระบบปฏิบัติการ

💡 ตัวอย่างเช่น การวนรอสถานะ อาจใช้ osDelay() (รอเวลามีหน่วยเป็นมิลลิวินาที) หรืออาจใช้ฟังก์ชันที่ออกแบบมาเฉพาะเพื่อการนี้ osThreadYield() เพื่อแจ้งให้ระบบปฏิบัติการว่าสามารถสลับทาสก์ได้ทันทีในจังหวะนี้

Task Name	Priority	Stack...	Entry Funct...	Code Gene...	Parameter	Allocation	Buffer Name	Control Blo.
runningLED	osPriorityNormal	128	runningLED...	Default	NULL	Dynamic	NULL	NULL
ultrasonic	osPriorityLow	128	ultrasonicT...	Default	NULL	Dynamic	NULL	NULL



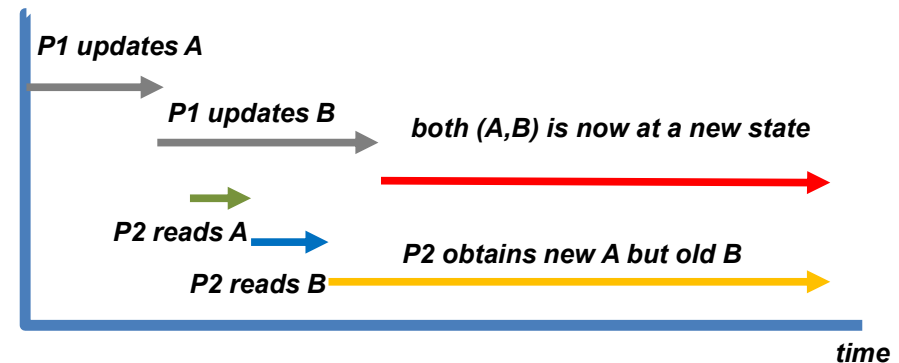
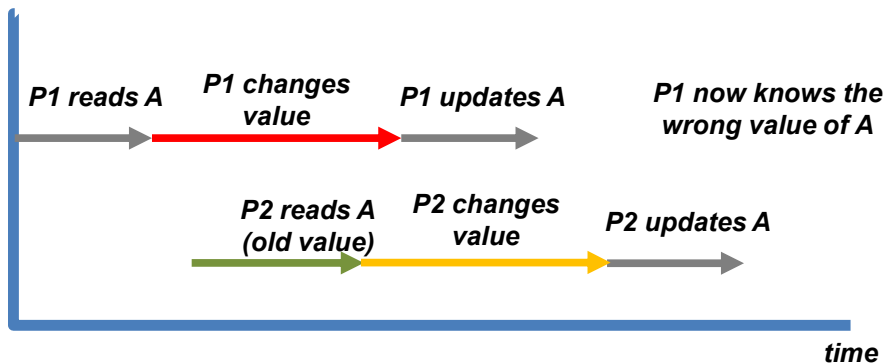
Asst.Prof. Thanwa SRIPRAMONG
PRESENTER

TODAY TOPIC IS

Multitasking Issues

สภาวะแข่งขัน (Race condition)

- 🌿 สถานการณ์ที่โปรเซส/เธรด มากกว่าหนึ่งตัวเข้าถึงทรัพยากรชุดเดียวกันพร้อมๆ กัน
 - 💡 มักจะพบปัญหาเมื่อโปรเซส/เธรดหนึ่งอ่าน ในขณะที่อีกโปรเซส/เธรดกำลังเขียน หรือกำลังเขียนพร้อมกัน
- 🌿 ผลที่เกิดขึ้น ทำให้โปรเซส/เธรด แต่ละตัวได้ข้อมูลที่ไม่เหมือนกัน
 - 💡 Data inconsistency
- 🌿 ตัวอย่างเช่นกรณีด้านล่าง



Asst.Prof. Thanwa SRIPRAMONG
PRESENTER

TODAY TOPIC IS

Multitasking Issues

สภาวะแข่งขัน (Race condition)

ลองพิจารณากรณีดังนี้

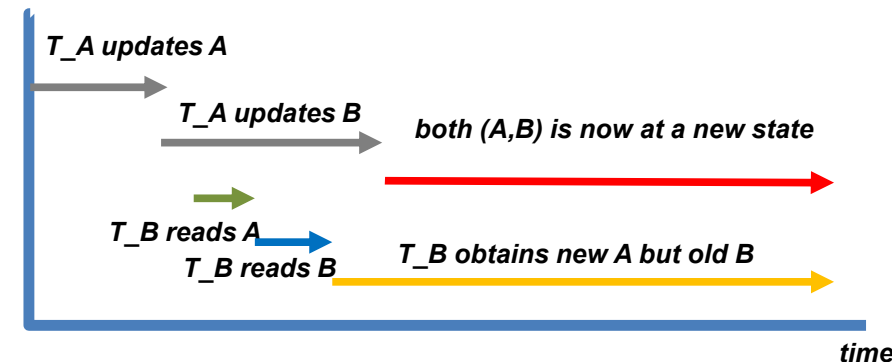
- สมมติว่ามีเธรด A อ่านค่าจากเซ็นเซอร์สองตัว ซึ่งใช้เวลาอ่านรวมประมาณ 2 ms แต่จะอ่านใหม่ทุกๆ ครั้งทุก 10 ms
- สมมติว่ามีเธรด B นำค่าจากเซ็นเซอร์ที่ได้จากเธรด A ไปใช้งานในทุกๆ 10 ms เช่นกัน

อาจเกิดสถานการณ์ที่

- เธรด A อัปเดตค่าจากเซ็นเซอร์ A ในตัวแปรร่วม แต่ยังไม่อัปเดตเซ็นเซอร์ B ในตัวแปรร่วมของรอบการอ่านปัจจุบัน
- เวลาเดียวกันนั้น เธรด B นำค่าจากตัวแปรร่วม A และ B ไปใช้งาน

สมมติว่าหากตรวจจับวัตถุได้จากเซ็นเซอร์ทั้งสองตัว ระบบจะหยุดทำงาน แต่ถ้าเซ็นเซอร์ตัวเดียวตรวจจับได้ จะหันหนีไปอีกทาง

- จะเกิดอะไรขึ้นหากพบกรณีที่กล่าวมาข้างต้น!!!



Asst.Prof. Thanwa SRIPRAMONG
PRESENTER

TODAY TOPIC IS

Multitasking Issues

Interlocking Mechanism in Multitasking OS

🌿 เป็นกลไกของระบบปฏิบัติการเพื่อใช้ล็อกทรัพยากรที่กำลังถูกจัดการ เพื่อไม่ให้โปรเซส/เธรดอื่นเข้าถึง (เป็นการชั่วคราว)

💡 จุดคำสั่งในบริเวณที่จัดการทรัพยากรรวมเรียกว่า ส่วนวิกฤติ (Critical section)

💡 กลไกตัวอย่าง mutex lock (mutex = mutual exclusion)

🌿 หลักการทั่วไปของการสร้างส่วนวิกฤติ

💡 ใช้ล๊อมชุดคำสั่งที่จัดการเข้าถึงทรัพยากรรวม (เช่นชุดตัวแปรที่ใช้ร่วมระหว่างโปรเซส/เธรด)

💡 เวลาในการประมวลผลภายในส่วนวิกฤติต้องสั้นที่สุดเท่าที่จะทำได้ ต้องไม่มีการรอคอยใดๆ ที่คำนวณเวลา รอไม่ได้

💡 การใช้ mutex lock มากกว่าหนึ่งตัวร่วมกันภายในส่วนวิกฤติ ต้องระมัดระวังการเกิดสภาวะติดตาย (deadlock)

🌈 รายละเอียดมากกว่านี้ศึกษาได้จากในเนื้อหาวิชาระบบปฏิบัติการ



Asst.Prof. Thanwa SRIPRAMONG
PRESENTER

TODAY TOPIC IS

Multitasking Issues



สรุปหัวข้อ

- 🌿 หลักการเบื้องต้นของการสร้างฟังก์ชันบริการการขัดจังหวะ (ISR) คือ จะต้องใช้เวลาทำงานให้สั้นที่สุด
- 🌿 หากมีเหตุการณ์การขัดจังหวะเดิมซ้ำๆ กันบ่อยหลายครั้งและไม่สามารถจัดการด้วย ISR ได้ทัน หากข้อมูลที่ต้องการไม่จำเป็นต้องได้รับครบทุกข้อมูล อาจใช้วิธีการ disable interrupt หรือเปลี่ยนไปเขียนแบบ polling ภายใต้เธรดของ RTOS แทน
- 🌿 การทำงานแบบหลายทาสก์ แบบ non-preemptive จะต้องรอให้แต่ละทาสก์/เธรด จบรอบการคำนวณด้วยตนเอง ในขณะที่แบบ preemptive นอกเหนือจากการทำงานในลักษณะแบบ non-preemptive แล้ว ยังมีโอกาสถูกสลับทาสก์ก่อนจบ CPU burst หากทำงานมานานเกินไป
- 🌿 RR (round-robin) เป็นวิธีการที่นิยมนำมาใช้เป็นพื้นฐานของการทำงานแบบ preemptive multitasking



Asst.Prof. Thanwa SRIPRAMONG
PRESENTER

TODAY TOPIC IS

Multitasking Issues



สรุปหัวข้อ (ต่อ)

- 🌱 การสร้างเธรดบน FreeRTOS ควรกำหนดลำดับความสำคัญของเธรดแต่ละตัวให้สูงเท่าที่จำเป็น
- 🌱 เธรดที่มีโอกาสที่ CPU burst time ยาว ควรกำหนดลำดับความสำคัญให้ต่ำกว่าเธรดอื่น เพื่อเปิดโอกาสให้เธรดอื่นสลับเข้ามาประมวลได้
- 🌱 ผู้พัฒนาสามารถช่วยการสลับทาส์กได้โดยใช้ `osThreadYield()` หรือ `osDelay()` ตรงจุดที่เห็นว่าสามารถเปิดโอกาสให้เธรดอื่นเข้ามาทำงานในจังหวะดังกล่าวได้
- 🌱 สภาวะแข่งขัน เป็นสภาวะที่หลายโปรเซส/เธรด เข้าถึง/เปลี่ยนแปลง ทรัพยากรร่วมกัน ทำให้เกิดการมองเห็นข้อมูล หรือสภาวะที่ไม่ตรงกัน
- 🌱 แก้ไขได้โดยการใช้ interlocking mechanism เพื่อบังคับให้เพียงโปรเซส/เธรด เดียวเท่านั้นเข้าถึง ทรัพยากรร่วม ทำงานให้เสร็จ จึงปล่อยให้โปรเซส/เธรด อื่นเข้าถึงได้



Asst.Prof. Thanwa SRIPRAMONG
PRESENTER

TODAY TOPIC IS

Multitasking Issues