应用注意事项

STM32CubeMX和STM32CubeIDE线程安全解决方案

介绍

本应用程序说明描述了用STM32CubeMX和STM32CubeIDE实现的线程安全解决方案，以确保对对C库函数的并发调用进行安全可靠的操作：

本应用程序说明中描述的线程安全问题和解决方案只与包含关键部分的函数相关，该函数操作于一个全局数据的单一实例，如malloc（）或免费（）。

应用程序说明没有提到重新进入，因为在线程安全的解决方案中没有锁定机制，保护用户免受与重新进入相关的问题。相反，用户有责任确保每个C库函数，比如strtok（），都用自己的线程上下文调用。

C库与STM32Cube生态系统中推广的工具链捆绑在一起，在默认情况下不受并发性的保护。这意味着，在多线程系统中，使用malloc（）、免费（）或任何其他功能隐式调用这些函数都是不安全的。这些多线程系统可以是裸金属系统或基于RTOS的系统。

本应用程序说明是启动线程安全解决方案的指导方针，并使用适当的线程安全策略创建和构建项目，以确保受支持的C库不受并发性的保护。所提出的RTOS策略及其实现仅适用于FreeRTOS™实时操作系统。

第2节介绍了线程安全问题和建议的解决方案，而第3节和第4节描述了在STM32CubeMX中生成的线程安全设置的实现，以及使用兼容的工具链生成的各种文件。第5节展示了常见的裸金属和FreeRTOS™示例，说明了如何通过采用线程安全的解决方案来确保安全的应用。

1 将军 消息

1.1 参考文献

STM32CubeMX用于STM32配置和初始化C代码生成（UM1718）

STM32CubeIDE用户指南（UM2609）(1)

1. 当前的应用程序说明补充了STM32CubeIDE用户指南中的线程安全向导部分。

1.2 兼容的工具链

本应用程序说明中介绍的STM32CubeMX和STM32CubeIDE线程安全解决方案可与以下工具链和软件工具的最低版本兼容：

意法半导体- STM32CubeMX： V6.3.0

意法半导体- STM32CubeIDE： v1.7.0

IAR系统®- IAR嵌入式工作台®（EWARM）： V8.50

•凯尔®-微控制器开发工具包臂®基于微控制器（MDK-ARM）： V4.73

笔记 *IAR系统公司是IAR系统AB公司拥有的注册商标。*

Arm和Keil是Arm有限公司（或其子公司）在美国和/或其他地方的注册商标。

2 螺纹安全 解决办法 背景

确保应用程序是线程安全的，这是整个嵌入式行业中的一个难题。随着更大内存嵌入式系统的引入以及RTOS使用率的增加，这些问题变得更加频繁。与STM32CubeMX-MDK-ARM、EWARM和STM32CubeIDE一起考虑的各种软件工具链都面临着在线程安全方面的类似问题。目前已经有一些用户贡献的解决方案，它们与STM32Cube生态系统并不完全兼容。本应用说明中提出的线程安全解决方案是由意法半导体公司开发的，以符合STM32立方体生态系统。

2. 线程安全问题的说明

术语线程和多线程同时适用于裸金属和基于RTOS的应用程序：

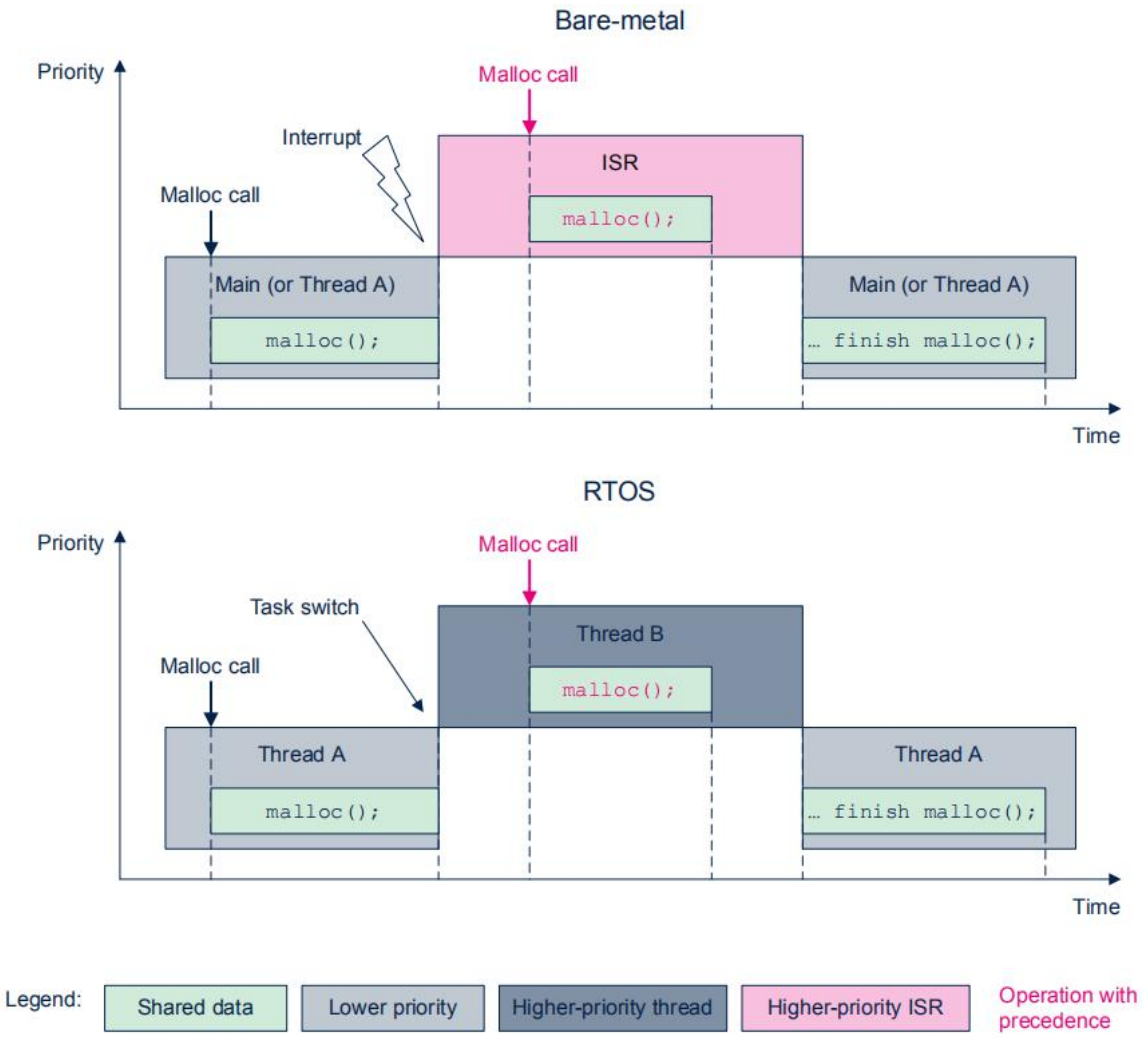
RTOS应用程序：在一个RTOS或一个ISR中的两个线程或任务（中断服务例程）

裸金属应用程序：主线程被ISR中断，ISR也被视为第二个执行线程

需要注意的是，线程安全问题并不只限于基于RTOS的应用程序；这个问题也存在于裸金属应用程序中，其中允许中断服务例程可以调用C库函程。线程安全问题可能会出现在多线程应用程序中，其中两个线程试图操作一个共享的全局数据实例，如malloc（）或免费（）。为了说明线程安全的问题，在图1中考虑了两个常见的用例，其中该问题可能在裸金属或RTOS应用程序中出现。应用程序代码（主代码或线程）调用malloc（）来分配一些内存。malloc（）包含一个关键部分，其中内存指针被更新为指向下一个空闲内存区域。如果在malloc（）执行更新此内存指针的关键部分时，允许CPU切换上下文，则存在malloc（）向两个线程返回相同的内存指针的风险。然后两个线程将数据写入同一个内存块，这意味着内存内容已损坏。

同样值得强调的是，C库可以进行不那么明显的调用（隐式调用），从而导致类似的问题。例如，printf（）可以调用malloc（）。对于最终用户，上面描述的线程安全问题很难进行调试。内存内容已损坏，但应用程序可能直到执行流之后才会崩溃。在这一点上，很难确定其根本原因是很久以前在应用程序的一个完全不同的模块中造成的内存损坏。

图1。用例示例

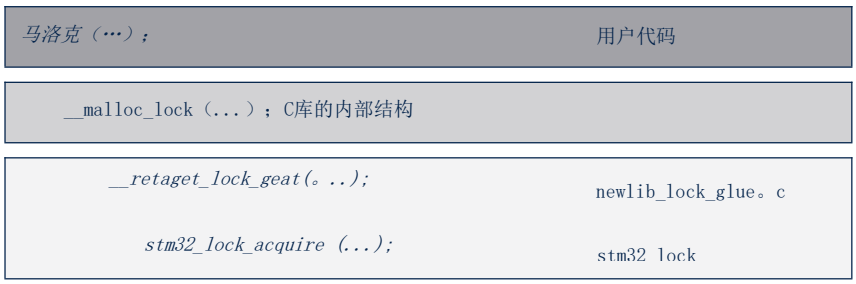

Bare-metal
Priority
Malloc call
Interrupt
ISR
malloc();
Malloc call
Main (or Thread A)
malloc();
Main (or Thread A)
… finish malloc();
Time
RTOS
Priority
Malloc call
Task switch
Thread B
malloc();
Malloc call
Thread A
malloc();
Thread A
… finish malloc();
Time
Legend:
Shared data
Lower priority
Higher-priority thread
Higher-priority ISR
Operation with precedence

2.2技术解决方案

提示：要进一步研究，请同时查看stm32_lock.h，以更好地理解。该文件可以由STM32CubeMX生成，如第4节所示。

C库是使用可延迟锁定来构建的。这意味着有引入线程安全问题风险的功能包含锁定和解锁钩子。在关键部分的进入时调用锁钩，在关键部分的出口时调用解锁钩。线程安全解决方案实现了所需的锁相关功能。图2显示了在调用malloc（）时，用户代码、C库和线程安全解决方案之间的简化交互。

图2。螺纹安全解决方案



线程安全解决方案是一种灵活的解决方案，它根据用户的需求为用户提供不同的线程安全策略。每个策略都代表了与锁相关的功能的一个独特的实现。

所有与锁相关的函数都在stm32_lock中实现了。h文件。这个文件是通用的，无论使用的工具链是EWARM、MDK-ARM还是STM32CubeIDE，该文件都适用。对于每个受支持的工具链（实际上是C库），都提供了一个专用的粘合剂文件。对于STM32CubeIDE，这对应于newlib_lock_glue.c。这个文件充当了C库中的钩子和文件stm32_lock.h中的实际锁实现之间的粘合剂。每种策略都有优缺点，这就是为什么用户必须在可能的策略中做出积极的选择。

线程安全解决方案支持了处理线程安全锁的五种策略。这些策略分为通用策略和RTOS依赖策略。

沙痂的一般策略

-策略#1：处理线程安全的用户定义解决方案。

-策略2：允许从中断中使用锁。

此实现通过在调用期间禁用所有中断来确保线程安全
马洛克（）。

-策略3：拒绝对中断使用锁。

此实现假设为单线程执行，并拒绝从ISR上下文中获取锁的任何尝试。

FreeRTOS™-基于上的策略

-策略#4：允许从中断中使用锁。

使用FreeRTOS™锁实现。该实现通过在调用malloc（）期间输入具有RTOS ISR功能的关键部分来确保线程安全性。这意味着线程安全是通过禁用低优先级中断和任务切换来实现的。然而，高优先级的中断是不安全的。

-策略5：拒绝对中断使用锁。

使用FreeRTOS™锁实现。这个实现通过在调用malloc（）期间暂停所有任务来确保线程的安全性。

3 STM32CubeMX 螺纹安全的 镶嵌

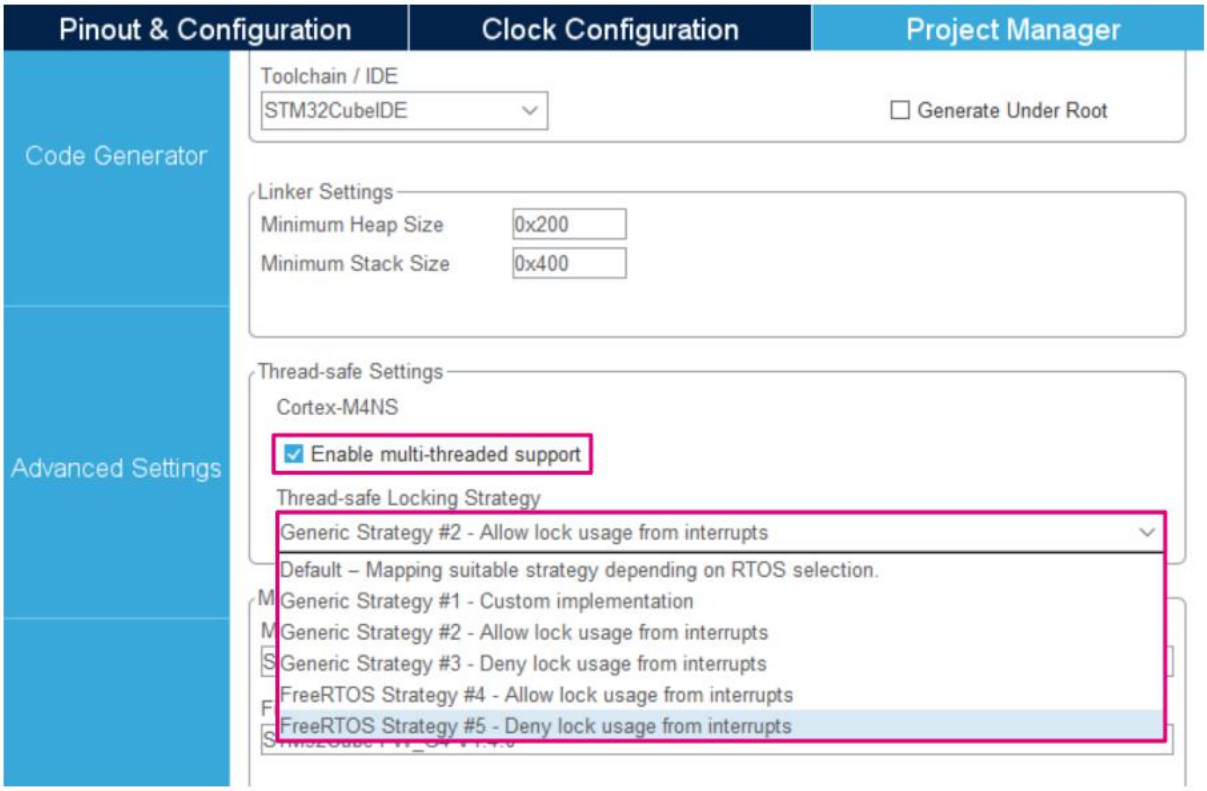
螺纹安全设置菜单

在“项目管理器设置”选项卡下，有“线程安全设置”部分，如图所示
图3。适用于所有STM32微控制器，支持的ide有： EWARM、MDK-ARM、
STM32立方体。
请注意，当选择另一个IDE时，线程安全部分将被禁用（灰色）。

启用复选框[启用多线程支持]，以便在[线程安全锁定策略]中提出了五种不同的线程安全策略的列表：

-默认值：根据RTOS的选择，映射合适的策略。
-通用策略#1：自定义实现。
-通用策略#2：允许从中断中使用锁。
-通用策略#3：拒绝对中断使用锁。
允许锁使用：依赖于FreeRTOS™。
-FreeRTOS策略#5：拒绝来自中断的锁使用。

图3。螺纹安全设置



螺纹安全策略选择

选择一个如图3所示的策略。

默认设置的使用会根据RTOS的选择来映射合适的策略。这有助于用户根据是否使用RTOS自动设置工具建议的推荐策略。所有策略都是可用的（无论配置，选择的微控制器，或FreeRTOS™可用性）。

-如果在STM32CubeMX中禁用了FreeRTOS™，则会自动选择策略#2。
-如果在STM32CubeMX中启用了FreeRTOS™，则会自动选择策略#4。

多核心项目

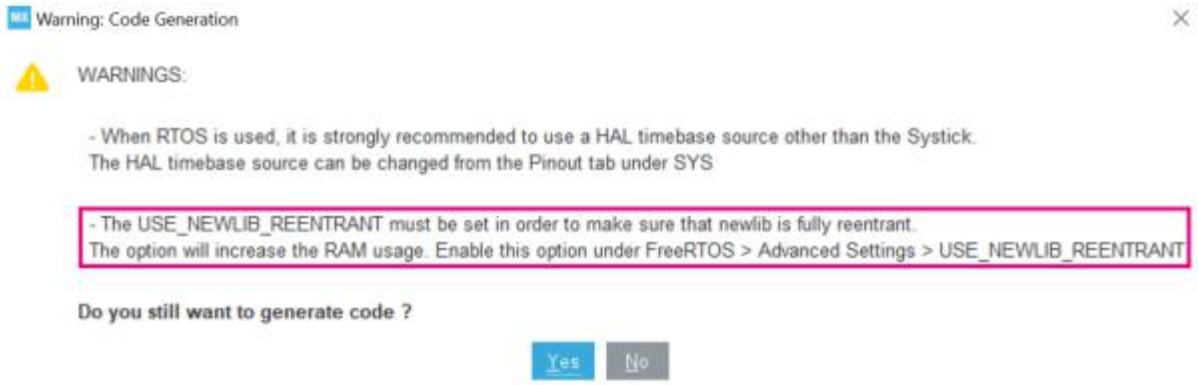
在多核项目中，每个核心项目都有一个线程安全的部分。

FreeRTOS™应用程序

在FreeRTOS™应用程序的情况下，FreeRTOS™根据_reent结构来管理对C库函数的重新入口支持，比如strtok（）。

如果在STM32CubeIDE中使用时使用STM32CubeMX配置项目，则configUSE_NEWLIB_REENTRANT参数必须等于1。STM32CubeMX检查此约束，如果不遵守此约束，则在代码生成步骤中显示一个警告消息（参见图4）。此标志允许为每个创建的任务分配新lib_reent结构；调度程序更新全局不纯指针指向激活任务的_reent结构。此设置仅防止在任务切换时重新进入，而不包括从ISR调用的重入功能。还请注意，重新进入支持增加了RAM的使用量。

图4。生成警告



4 项目 构造 和 文件 构造 使用 软件 工具链

必须在项目结构中添加三个文件：

• stm32_lock .h：定义了不同的锁定策略。

xxx_lock_glue.c（取决于工具链）：实现必要的锁定粘合剂来保护C库函数和C++中的本地静态对象的初始化。

• stm32_lock_user .h：用于实现自定义策略的用户文件。

用户可以手动更新stm32_lock_user.h，它在代码再生期间由STM32CubeMX保存。

4. EWARM toolchain

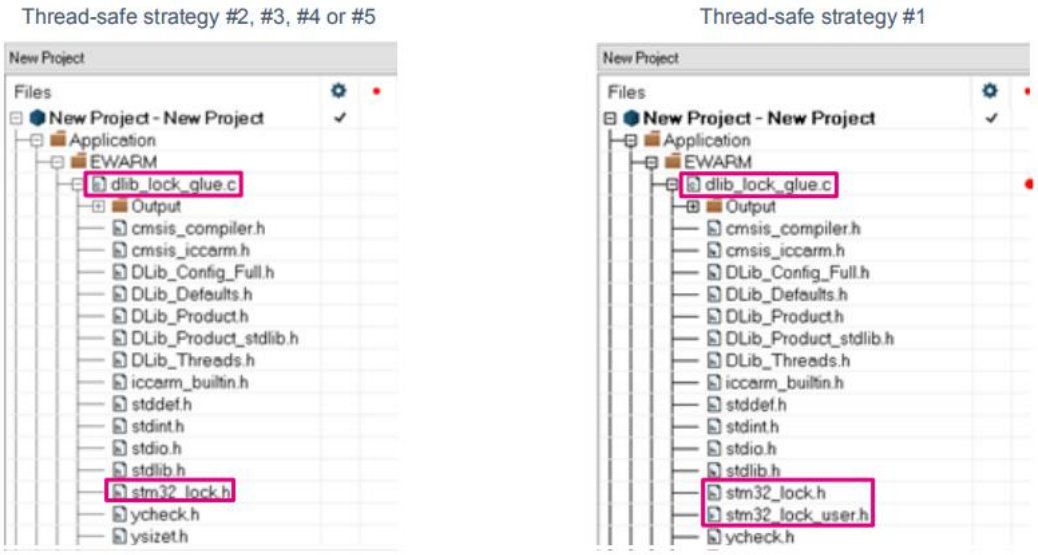
单芯工程

对于单核项目：

如果用户选择策略#2、#3、#4或#5，则将生成两个文件： stm32_lock。h和dlib_lock_glue.c（见图5）

如果用户选择策略#1，则将生成三个文件： stm32_lock。h, dlib_lock_glue .c和stm32 _lock_user。h（请参见图5）

图5。EWARM单核项目配置



多核心项目

在多核项目中，相同的文件(stm32_lock.h和dlib_lock_glue。c)在每个核心项目中被引用，每个核心项目使用一个单独的文件（stm32_lock_user.h）。

4. MDK-ARM工具链

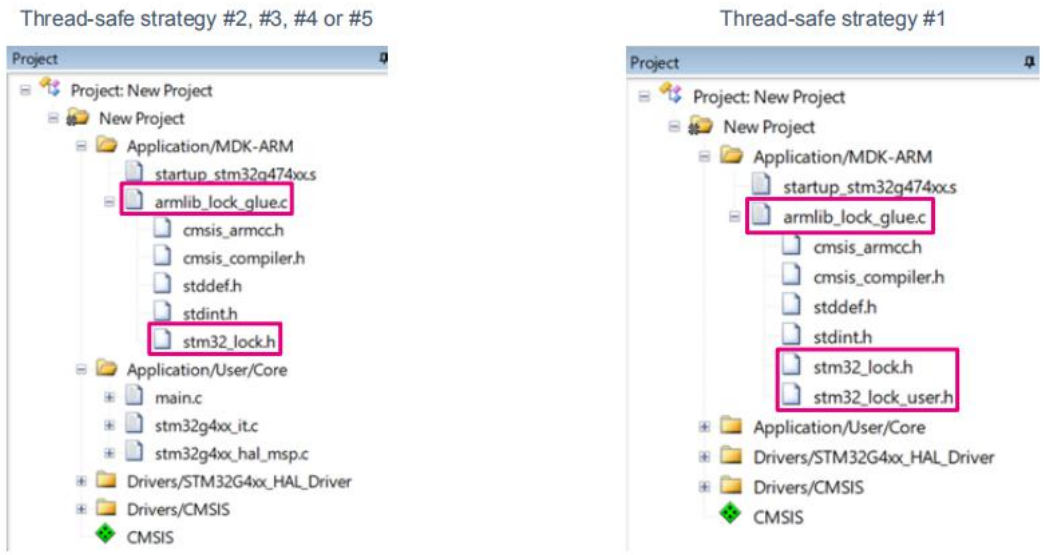
单芯工程

对于单核项目：

如果用户选择策略#2、#3、#4或#5，则将生成两个文件： stm32_lock。h和armlib_lock_gl。c（请参见图6）

如果用户选择策略#1，则将生成三个文件： stm32_lock。h，armlib_lock_glue .c和stm 32_lock_user.h（见图6）

图6。MDK-ARM单核项目配置



多核心项目

在多核项目中，相同的文件(stm32_lock.h和armlib_lock_glue。c)在每个核心项目中被引用，每个核心项目使用一个单独的文件（stm32_lock_user.h）。

4. STM32CubeIDE toolchain

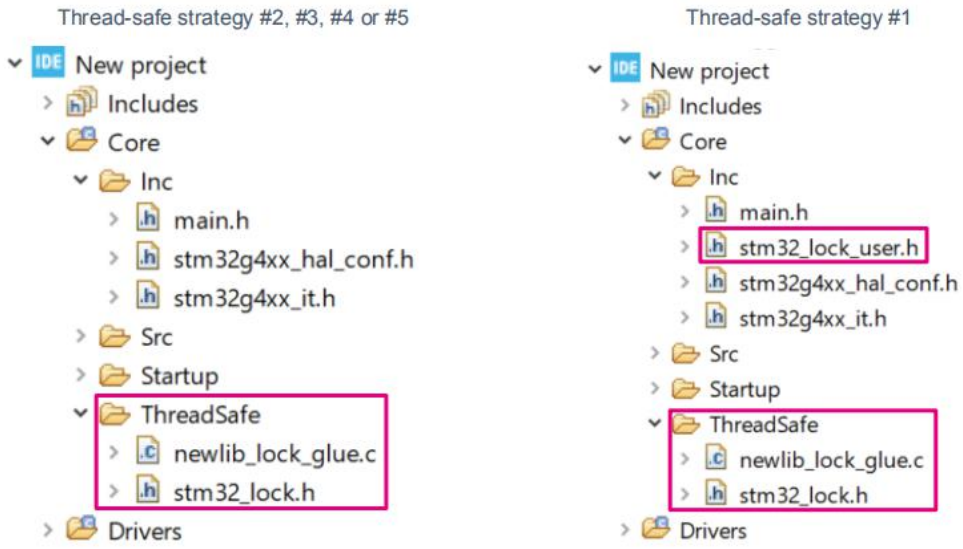
单芯工程

对于单核项目：

如果用户选择策略#2、#3、#4或#5，则将生成两个文件： stm32_lock。h和newlib_lock_gl。c（请参见图7）

如果用户选择策略#1，则将生成三个文件： stm32_lock。h，newlib_lock_胶水。c和stm 32_lock_user.h（见图7）

图7。STM32CubeIDE单核项目配置



多核心项目

在多核项目中，相同的文件(stm32_lock.h和newlib_lock_glue。c)在每个核心项目中被引用，每个核心项目使用一个单独的文件（stm32_lock_user.h）。

空项目

在STM32CubeIDE空项目的情况下，可以生成一个线程安全的解决方案。有关更多详细信息，请参阅STM32CubeIDE用户指南（UM2609）中的空项目和CDT™项目的线程安全向导部分，该指南可在www.st.com上获得。

5 使用 情况 例子

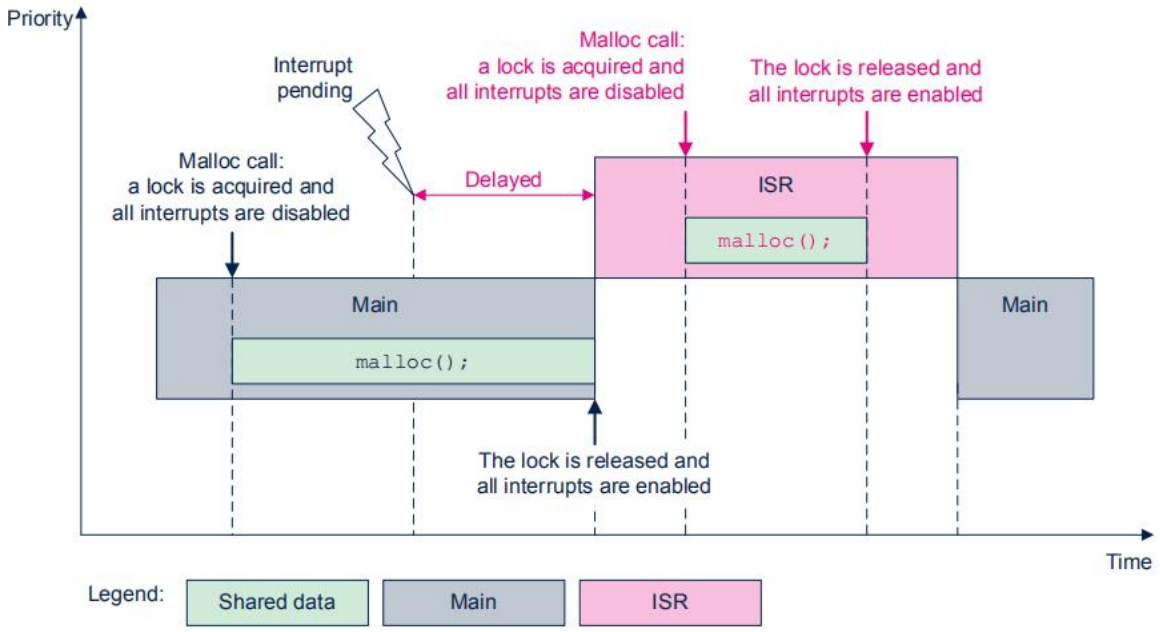
以支持图1中描述的用例。用例，本章详细介绍了一些显示裸金属和FreeRTOS™策略部署的示例。每种策略都提供锁实现有其优点和缺点。

5.1 裸金属应用

5.1.1 使用策略#2的用例

该策略通过在获取锁时暂时禁用所有中断来确保线程安全（参见图8）。中断保持未决，并且不能抢占当前活动的执行上下文。当malloc（）的执行完成后，将释放锁，从而重新启用中断。因此，CPU能够再次切换上下文并执行中断。如果ISR调用malloc（），它就会接受一个锁，完成其执行，并释放锁。应用程序从ISR重新进入是安全的，共享数据没有损坏。然而，这种策略的副作用是中断被延迟。

图8。使用策略#2的用例

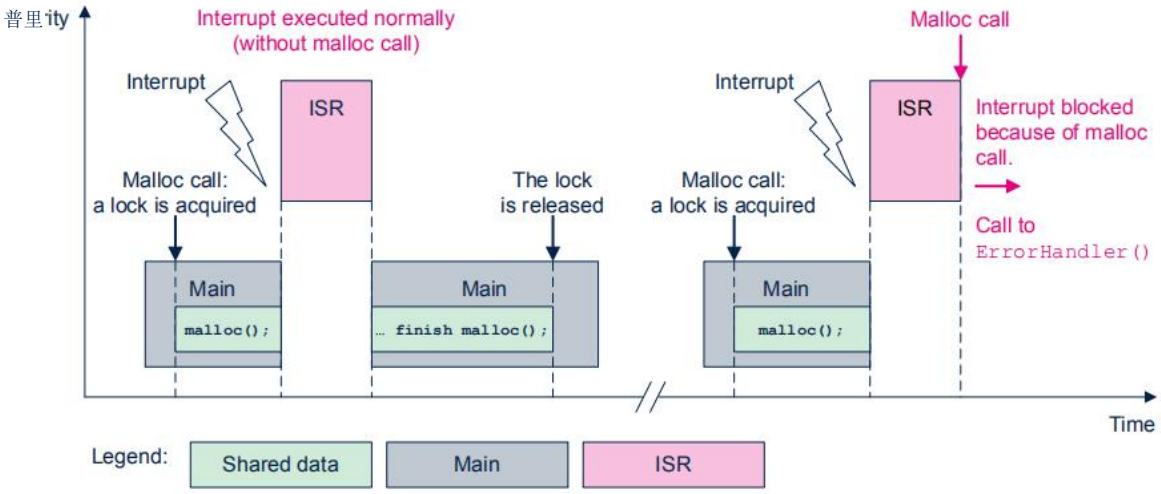


5.1. 使用策略#3的用例

该策略允许在获取锁（参见图9）时从ISR进行上下文切换，并且中断不会延迟。但是使用这种策略，不可能从ISR上下文中获得锁。因此，当C库函数试图从ISR上下文中获取锁时，该尝试会被拒绝，CPU也会卡在错误处理程序（）中。

在图9的左边，主（）调用malloc（）。malloc（）执行中断，但ISR不调用malloc（）；因此，没有尝试从ISR上下文获取锁。没有任何数据被损坏，因为malloc（）可以在从ISR上下文返回后完成关键部分。
在图9的右边，主（）调用malloc（）。malloc（）的执行被中断，ISR调用malloc（）；因此，就会尝试从ISR上下文中获取锁。不允许从ISR获取锁，并且应用程序挂起在错误处理程序（）中。其目的是向开发人员发送一个明确的信号，即C库不能以这种方式使用。通过使开发人员意识到在ISR上下文中使用C库函数的危险情况，该应用程序被认为是安全的。

图9。使用策略#3的用例



5.2 FreeRTOS™应用

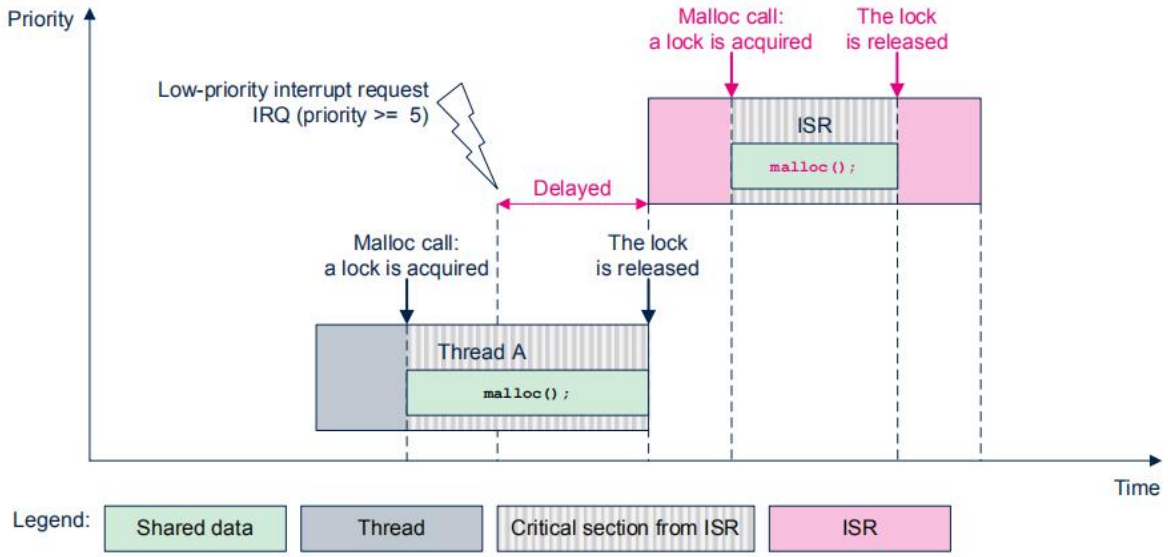
在一个基于RTOS的应用程序中，关于对C库函数的并发调用，有三个可能的来源：
沙痂低优先级中断
-用于时间敏感较低的操作
-用于勾选RTOS
-用于在FreeRTOS™任务之间执行切换
高优先级中断：可能需要在应用程序中进行，例如执行时间关键的操作
沙痂任务交换机

5.2.1 使用策略#4的用例

该策略通过在调用malloc（）期间输入具有RTOS ISR能力的临界部分来确保线程的安全性
通过宏taskENTER_CRITICAL_FROM_ISR。已实现了taskENTER_CRITICAL_FROM_ISR宏
略有不同，这取决于哪个皮质®M是该项目的核心目标。当获得了锁时，
malloc（）进入一个关键部分，因此低优先级的中断和任务开关被禁用。然而
高优先级中断仍然可以以并发C库函数调用的不安全为代价发生。
默认情况下，此实现支持两级递归锁定。递归级别的数量是可配置的。

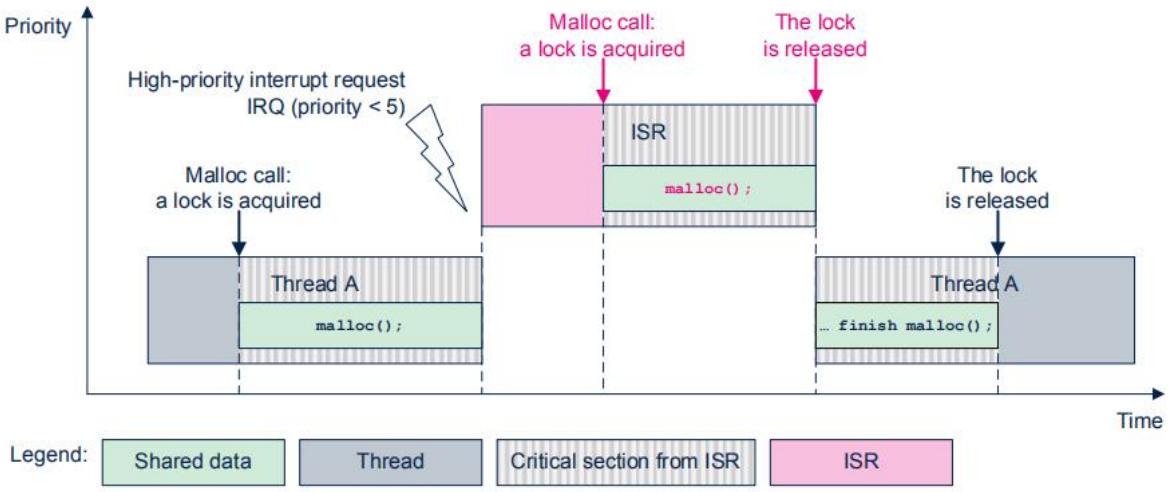
如图10所示，低优先级中断（高优先级值）会延迟，直到taskEXIT_CRITICAL_FROM_ISR完成并释放锁定。因此，malloc（）受到其他并发调用的保护，共享数据没有损坏。

Figure 10. Use case with strategy #4 (low-priority interrupt)



在高优先级中断的情况下（参见图11），将执行malloc（）调用，因此可能会损坏共享的数据。

图11。策略#4的用例（高优先级中断）

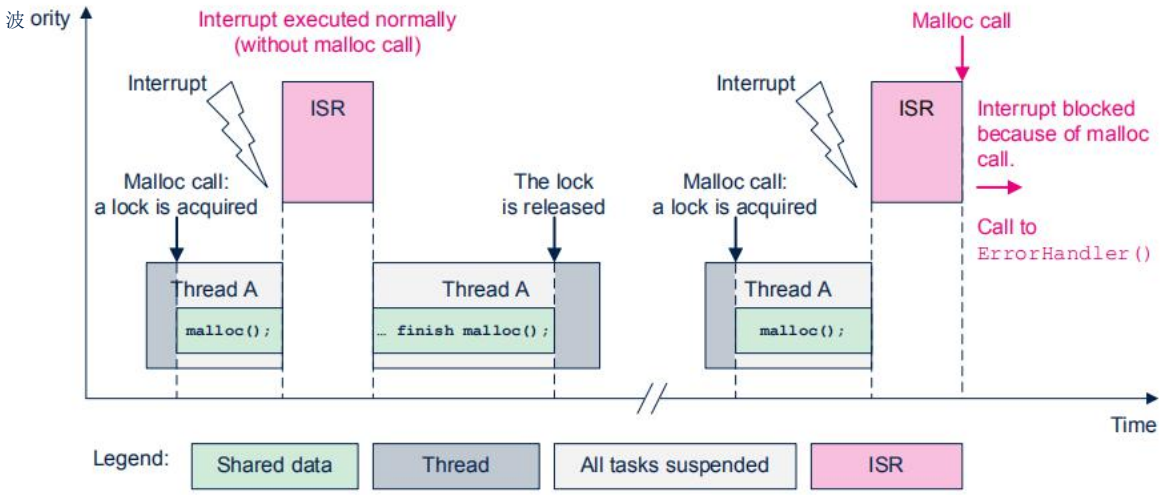


注意：IRQ优先级阈值为5，这只是一个例子。用户有责任配置优先级屏蔽。

5.2. 使用策略#5的用例

该策略通过在malloc（）调用期间暂停所有任务，但不启用中断（参见图12）。当malloc（）获取锁时，只有释放锁后才能发生任务切换。但是，允许中断来切换执行。但如果ISR试图获得一个锁
从ISR上下文中，应用程序将被困在错误处理程序（）中并挂起。因此，通过使开发人员意识到C库函数的危险使用，该应用程序在ISR上下文中也被认为是安全的。这种方法类似于策略#3。

图12。使用策略#5的用例



6 钥匙 外卖餐馆

不建议使用可能需要同步的、同时来自主应用程序（或线程）和ISR的C库函数。

大多数与线程安全问题相关的支持请求都涉及到在FreeRTOS™应用程序中使用malloc（）。线程安全的解决方案保护malloc（）和其他C库

在执行关键部分时，关于并发调用的函数。此外，启用configUSE_NEWLIB_REENTRANT标志可以确保任何像strtok（）这样具有重新入口的函数都可以跨多个线程安全地使用，因为每个任务都得到一个唯一的_reent结构。但是，如果从ISR中调用strtok（），用户就不会受到保护。因此，用户必须小心来自isr的C库调用，并手动管理_reent结构实例。

用户必须小心，不要隐式调用线程安全函数。例如，printf（）可以在某些情况下调用malloc（）。

针对newlib函数的锁的实现会暂时禁用中断（策略#2和#4），从而影响应用程序的实时行为。在采用锁定机制之前，请仔细考虑在应用程序中禁用中断的影响。

修订历史记录

Table 1. **Document revision history**

Date	Revision	Changes
30-Nov-2021	1	Initial release.

内容

表列表

数字列表

重要的注意事项-请仔细阅读

意法半导体公司及其子公司（“ST”）保留在任何时候在不另行通知的情况下对ST产品和/或本文件进行更改、更正、增强、修改和改进的权利。买方在下单前应获取有关ST产品的最新相关信息。ST产品根据订单确认时ST的销售条款和条件进行销售。

买方全权负责ST产品的选择、选择和使用，ST不承担应用协助或买方产品设计的责任。

ST在此不授予任何知识产权的明示或默示许可。

转售与本协议规定的条款不同的ST产品，将使ST对该产品给予的任何保证无效。

ST和ST标志是ST的商标。有关ST商标的更多信息，请参考www。st.com/trademarks.所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档任何以前版本中提供的信息。