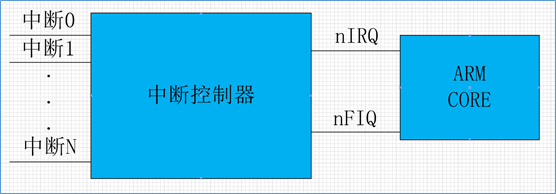
**实验五 时钟Tick**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab5/index.html#tick)

计科2302 王任杰 202308010212

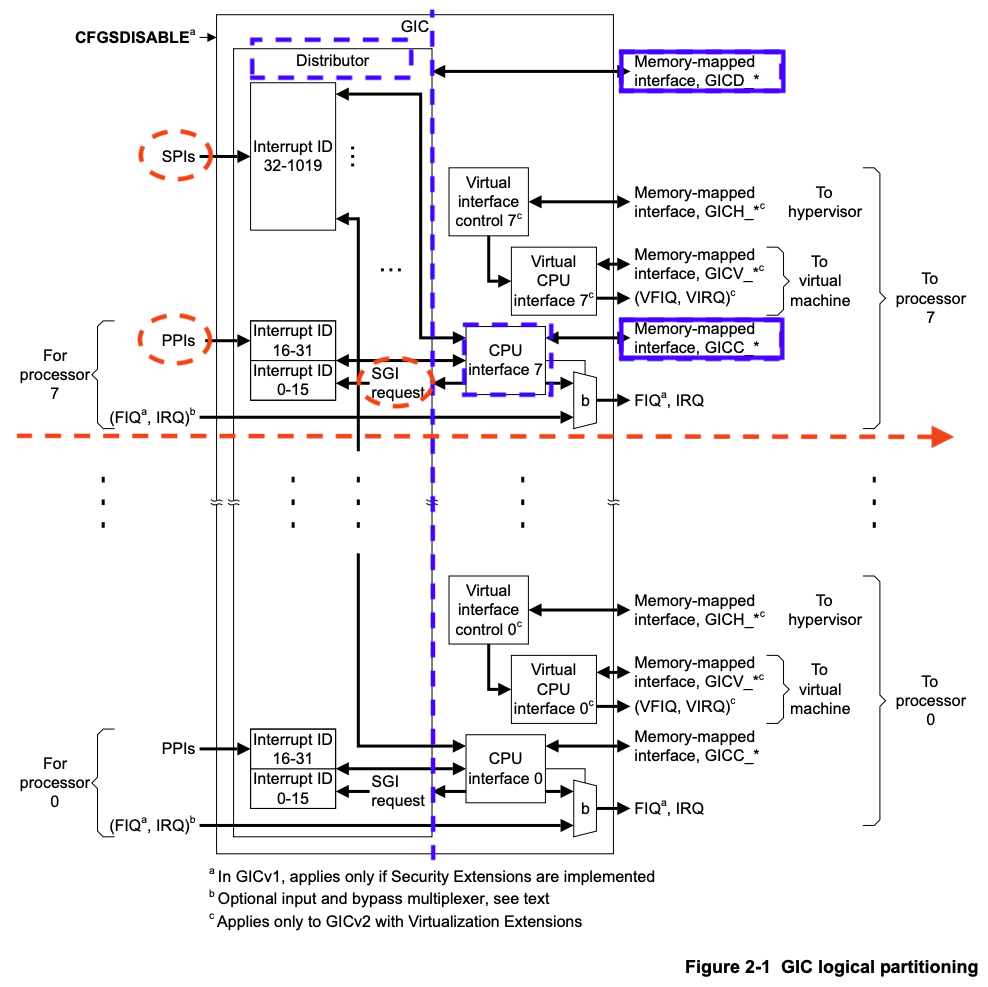
**Arm的中断系统[ℑ](https://os2024lab.readthedocs.io/zh-cn/latest/lab5/index.html" \l "arm" \o "此标题的永久链接)**

中断是一种硬件机制。借助于中断，CPU可以不必再采用轮询这种低效的方式访问外部设备。将所有的外部设备与CPU直接相连是不现实的，外部设备的中断请求一般经由中断控制器，由中断控制器仲裁后再转发给CPU。如下图所示Arm的中断系统。



其中nIRQ是普通中断，nFIQ是快速中断。 Arm采用的中断控制器叫做GIC，即general interrupt controller。gic包括多个版本，如GICv1（已弃用），GICv2，GICv3，GICv4。简单起见，我们实验将选用GICv2版本。

为了配置好gicv2中断控制器，与pl011串口一样，我们需要阅读其技术参考手册。访问Arm官网在 [这里](https://developer.arm.com/documentation/ihi0048/latest) 下载ARM Generic Interrupt Controller Architecture Specification - version 2.0 的pdf版本。



从上图（来源于ARM Generic Interrupt Controller Architecture Specification - version 2.0中的Chapter 2 GIC Partitioning）可以看出：

* GICv2 最多支持8个核的中断管理。
* GIC包括两大主要部分（由图中蓝色虚竖线分隔，Distributor和CPU Interface由蓝色虚矩形框标示），分别是：
  + Distributor，其通过GICD\_开头的寄存器进行控制（蓝色实矩形框标示）
  + CPU Interface，其通过GICC\_开头的寄存器进行控制（蓝色实矩形框标示）
* 中断类型分为以下几类（由图中红色虚线椭圆标示）：
  + SPI：（shared peripheral interrupt），共享外设中断。该中断来源于外设，通过Distributor分发给特定的core，其中断编号为32-1019。从图中可以看到所有核共享SPI。
  + PPI：（private peripheral interrupt），私有外设中断。该中断来源于外设，但只对指定的core有效，中断信号只会发送给指定的core，其中断编号为16-31。从图中可以看到每个core都有自己的PPI。
  + SGI：（software-generated interrupt），软中断。软件产生的中断，用于给其他的core发送中断信号，其中断编号为0-15。
  + virtual interrupt，虚拟中断，用于支持虚拟机。图中也可以看到，因为我们暂时不关心，所以没有标注。
  + 此外可以看到(FIQ, IRQ)可通过b进行旁路，我们也不关心。如感兴趣可以查看技术手册了解细节。

此外，由ARM Generic Interrupt Controller Architecture Specification - version 2.0 (section 1.4.2)可知，外设中断可由两种方式触发：

* edge-triggered: 边沿触发，当检测到中断信号上升沿时中断有效。
* level-sensitive：电平触发，当中断源为指定电平时中断有效。

因为soc中，中断有很多，为了方便对中断的管理，对每个中断，附加了中断优先级。在中断仲裁时，高优先级的中断，会优于低优先级的中断，发送给cpu处理。当cpu在响应低优先级中断时，如果此时来了高优先级中断，那么高优先级中断会抢占低优先级中断，而被处理器响应。

由ARM Generic Interrupt Controller Architecture Specification - version 2.0 (section 3.3)可知，GICv2最多支持256个中断优先级。GICv2中规定，所支持的中断优先级别数与GIC的具体实现有关，如果支持的中断优先级数比256少（最少为16），则8位优先级的低位为0，且遵循RAZ/WI（Read-As-Zero, Writes Ignored）原则。

**GICv2初始化[ℑ](https://os2024lab.readthedocs.io/zh-cn/latest/lab5/index.html" \l "gicv2" \o "此标题的永久链接)**

由下图中virt.dts中intc和timer的部分

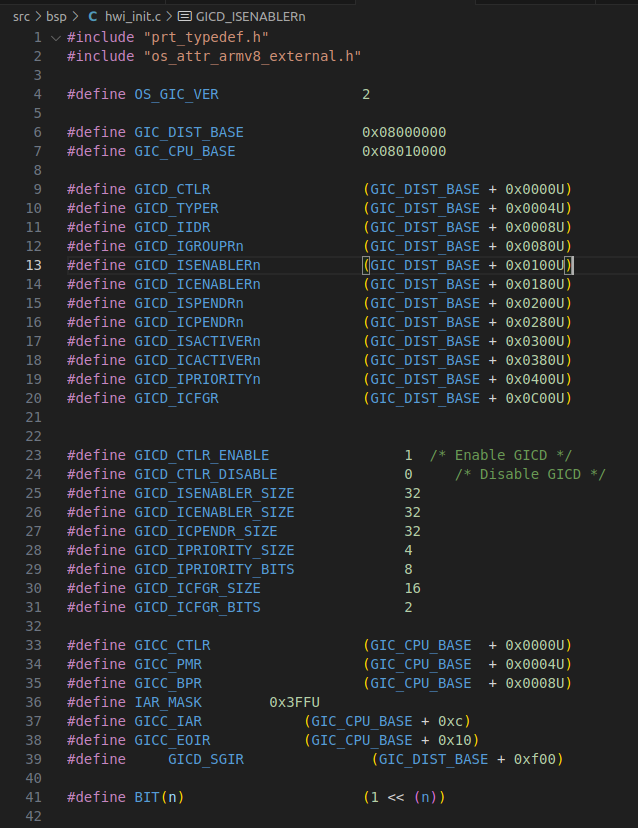


并结合kernel.org中关于 [ARM Generic Interrupt Controller](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm%2Cgic.txt) 和 [ARM architected timer](https://www.kernel.org/doc/Documentation/devicetree/bindings/arm/arch_timer.txt) 的devicetree的说明可知：

* intc中的 reg 指明GICD寄存器映射到内存的位置为0x8000000，长度为0x10000， GICC寄存器映射到内存的位置为0x8010000，长度为0x10000
* intc中的 #interrupt-cells 指明 interrupts 包括3个cells。[第一个文档](https://www.kernel.org/doc/Documentation/devicetree/bindings/interrupt-controller/arm%2Cgic.txt) 指明：第一个cell为中断类型，0表示SPI，1表示PPI；第二个cell为中断号，SPI范围为[0-987]，PPI为[0-15]；第三个cell为flags，其中[3:0]位表示触发类型，4表示高电平触发，[15:8]为PPI的cpu中断掩码，每1位对应一个cpu，为1表示该中断会连接到对应的cpu。
* 以timer设备为例，其中包括4个中断。以第二个中断的参数 0x01 0x0e 0x104 为例，其指明该中断为PPI类型的中断，中断号14， 路由到第一个cpu，且高电平触发。但注意到PPI的起始中断号为16，所以实际上该中断在GICv2中的中断号应为16 + 14 = 30。

阅读ARM Generic Interrupt Controller Architecture Specification - version 2.0，在其Chapter 4 Programmers’ Model部分有关于GICD和GICC寄存器的描述，以及如何使能Distributor和CPU Interfaces的方法。

新建 src/bsp/hwi\_init.c 文件，初始化 GIC



在 hwi\_init.c 中 OsHwiInit 函数实现 GIC 的初始化，此外还提供了其他函数实现开关指定中断、设置中断属性、确认中断和标记中断完成等功能。

**注意**

你需要参照 OsGicIntSetPriority 等函数实现 OsGicEnableInt 和 OsGicDisableInt 函数。

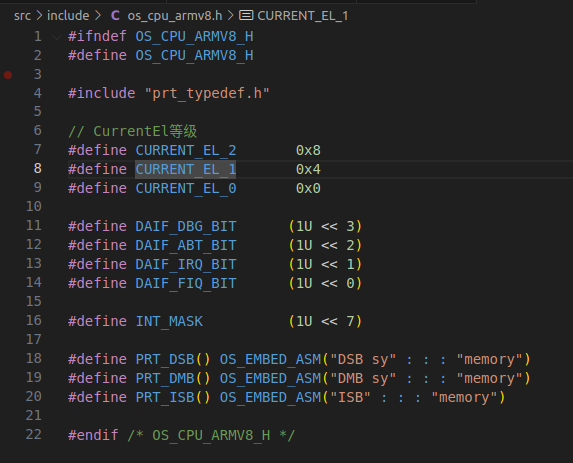
**使能时钟中断**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab5/index.html#id3)

新建 src/include/prt\_config.h

1*/\* Tick中断时间间隔，tick处理时间不能超过1/OS\_TICK\_PER\_SECOND(s) \*/*

2#define OS\_TICK\_PER\_SECOND 1000

新建 src/include/os\_cpu\_armv8.h。

****

新建 src/bsp/timer.c 文件，对定时器和对应的中断进行配置



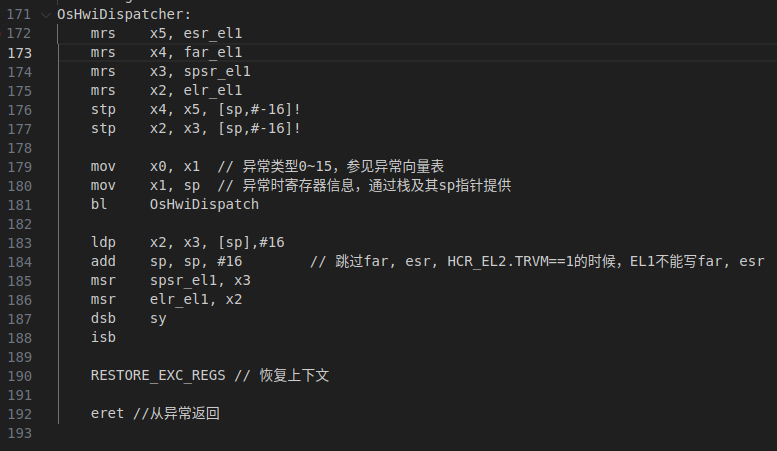
**时钟中断处理**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab5/index.html#id4)

将 prt\_vector.S 中的 EXC\_HANDLE 5 OsExcDispatch 改为 EXC\_HANDLE 5 OsHwiDispatcher，表明我们将对 IRQ 类型的异常（即中断）使用 OsHwiDispatcher 处理。

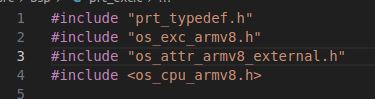
**提示**

需修改为 EXC\_HANDLE 5 OsHwiDispatcher ，否则还是 OsExcDispatch 函数处理，仅会输出 “Catch a exception.” 信息

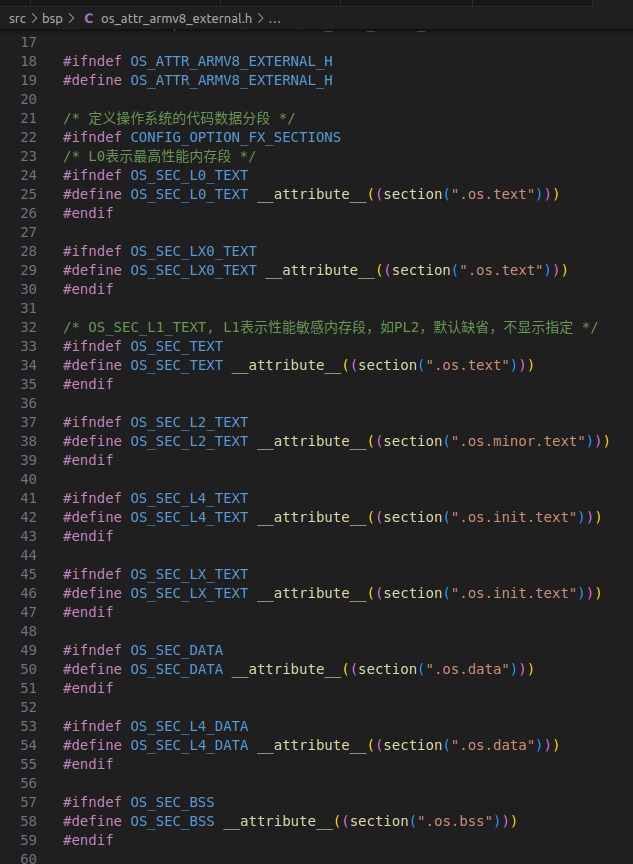




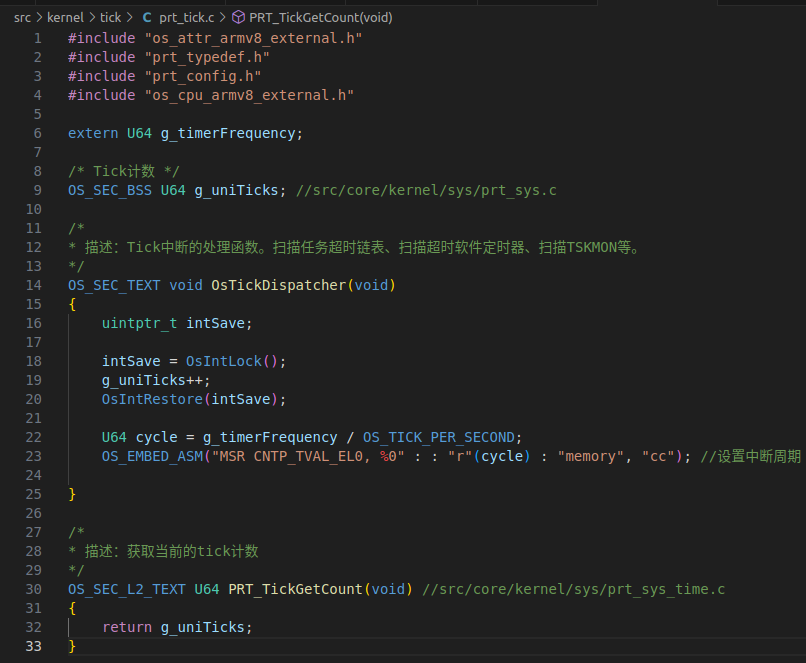
在 prt\_exc.c 中引用头文件 os\_attr\_armv8\_external.h ， os\_cpu\_armv8.h ， OsHwiDispatch 处理 IRQ 类型的中断。



src/bsp/os\_attr\_armv8\_external.h 头文件

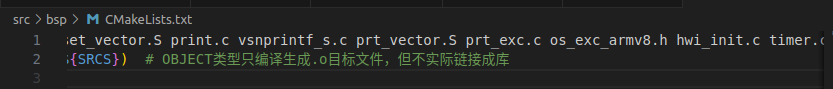


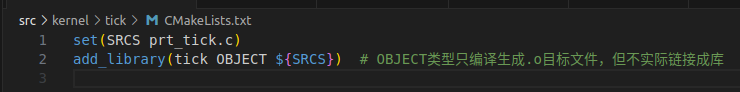
新建 src/kernel/tick/prt\_tick.c 文件，提供 OsTickDispatcher 时钟中断处理函数。



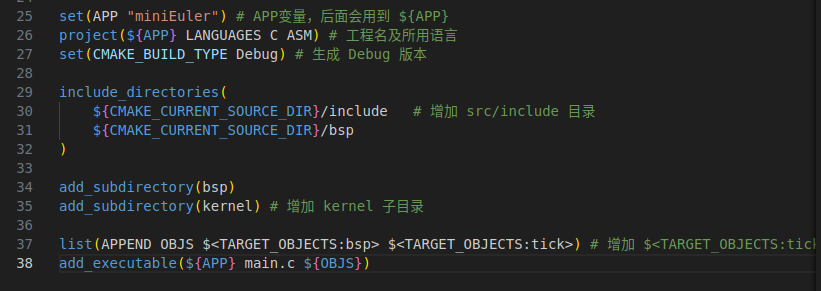
注意需将 hwi\_init.c timer.c prt\_tick.c 等文件加入构建系统。

**提示**



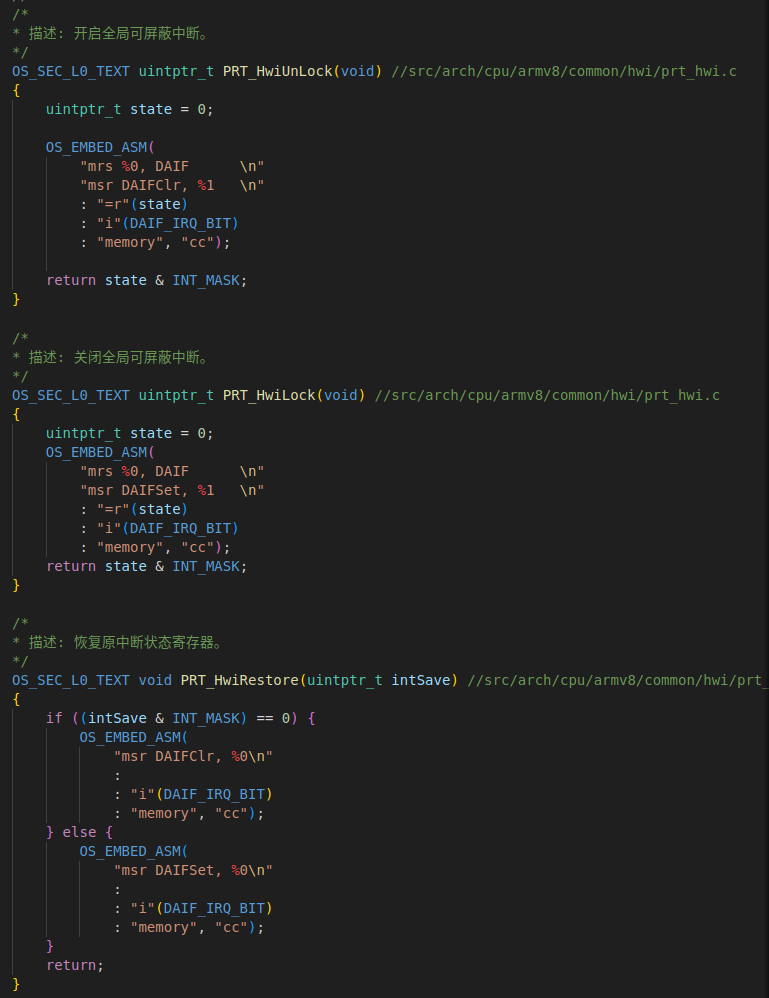




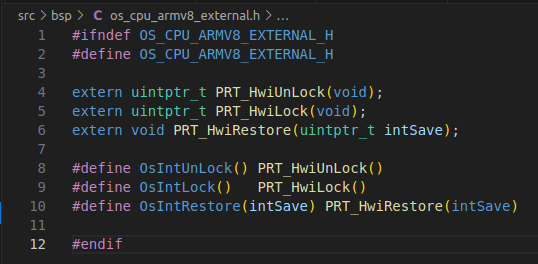


后续实验中若新增文件加入构建系统不再赘述，请参照此处。

* 在 OsTickDispatcher 中调用了 OsIntLock 和 OsIntRestore 函数，这两个函数用于关中断和开中断。简单起见，将其放入 prt\_exc.c 中。

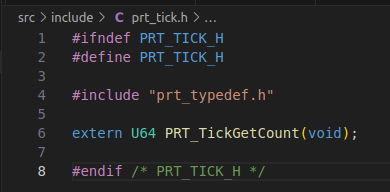


头文件 src/bsp/os\_cpu\_armv8\_external.h

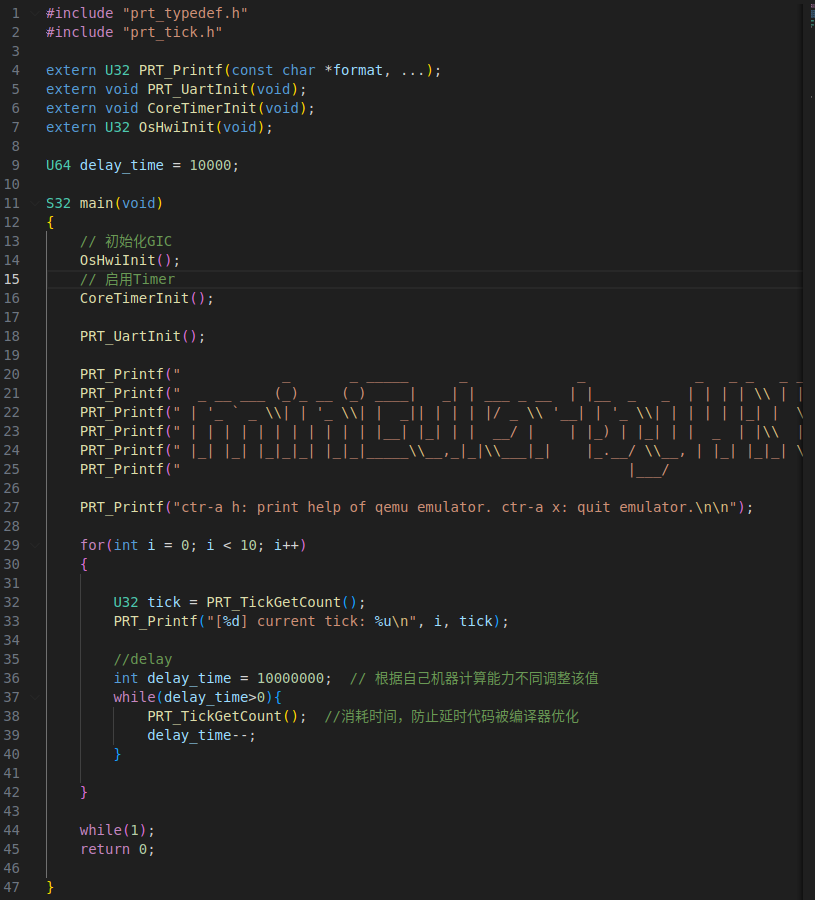


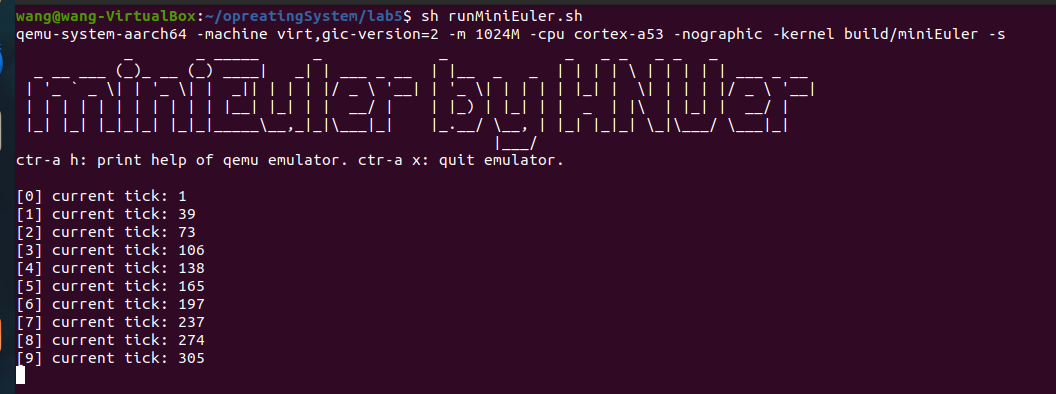
**读取系统Tick值[ℑ](https://os2024lab.readthedocs.io/zh-cn/latest/lab5/index.html" \l "id6" \o "此标题的永久链接)**

新建 prt\_tick.h，声明 Tick 相关的接口函数.



main.c 修改为：





**lab5 作业**

**作业1**

实现 hwi\_init.c 中缺失的 OsGicEnableInt 和 OsGicDisableInt 函数。

先用中断号除以每个寄存器可管理的中断数，从而确定该中断处于哪个寄存器；再乘以寄存器的大小（一般是 4 字节），这样就能得到该寄存器相对于基地址的偏移量

通过中断号对每个寄存器可管理的中断数取模，得到中断在对应寄存器里的具体位位置

