扩展练习 Challenge：

说明语句local_intr_save(intr_flag);....local_intr_restore(intr_flag);是如何实现开关中断的?

中断使能与禁止函数 intr_enable 和 intr_disable 是两个用来控制中断使能的函数：

- intr_enable: 使能中断，通过设置 SSTATUS_SIE位为1来恢复中断。 void intr_enable(void) { set_csr(sstatus, SSTATUS_SIE); // 设置 SSTATUS 寄存器中的 SIE 位 }
- intr_disable: 禁止中断，通过清除 SSTATUS_SIE 位来禁用中断。 void intr_disable(void) { clear_csr(sstatus, SSTATUS_SIE); // 清除 SSTATUS 寄存器中的 SIE 位 } 2.2 保存与恢复中断状态 通过函数 __intr_save 和 __intr_restore 来保存和恢复中断的状态。
- __intr_save: 判断当前中断使能状态，如果中断使能 (SSTATUS_SIE == 1)，则调用 intr_disable 禁止中断，并返回 1，表示中断被禁用；如果中断已经禁止，直接返回 0。 static inline bool __intr_save(void) { if (read_csr(sstatus) & SSTATUS_SIE) { // 如果 SSTATUS_SIE 位为 1 intr_disable(); // 禁止中断 return 1; // 返回 1，表示中断已禁用 } return 0; // 返回 0，表示中断未禁用 }
- __intr_restore: 根据 intr_flag 的值来判断是否需要恢复中断。如果 intr_flag 为 1，表示中断之前被禁用过，调用 intr_enable 恢复中断；如果 intr_flag 为 0，说明中断原本就是禁用状态，无需做任何操作。

在进行进程切换的时候，需要避免出现中断干扰这个过程，所以需要在上下文切换期间清除 IF 位屏蔽中断，并且在进程恢复执行后恢复 IF 位。

- 该语句的作用是关闭中断，使得在这个语句块内的内容不会被中断打断，是一个原子操作；
- 这就使得某些关键的代码不会被打断，从而不会引起不必要的错误；
- 比如说在 proc_run 函数中，将 current 指向了要切换到的线程，但是此时还没有真正将控制权转移过去，如果在这个时候出现中断打断这些操作，就会出现 current 中保存的并不是正在运行的线程的线程控制块，从而出现错误。

另，问题回答的补充：

```
define local_intr_save(x)
do {
x = __intr_save();
} while (0)
```

使用 do { } while(0) 语法来封装操作。

这样可以确保：

1. 避免潜在的警告：在某些情况下，宏定义需要包含一个语句块，即使宏没有具体实现的内容。

2. 创建独立的语句块：有时我们需要在宏中定义局部变量或者进行复杂操作，这种封装能确保代码清晰且无误。
3. 确保宏操作的原子性：当宏出现在条件语句之后时，do { } while(0) 确保宏作为一个独立的整体被执行。