



StaRT RTOS 移植指南 (Porting Guide)

目标：将 StaRT 运行在新的 Cortex-M 或其它架构 MCU 上。核心点：上下文切换、栈初始化、关中断、SysTick 时基、最低限度的汇编支持函数、可选的位操作 (`_s_ffs`)。

1. 必须实现/适配的接口清单

分类	函数	说明
上下文切换	<code>s_normal_switch_task(prev, next)</code>	保存当前线程 PSP，恢复下一线程 PSP，触发 PendSV/直接切换
首次切换	<code>s_frist_switch_task(next)</code>	初始进入第一个线程
栈初始化	<code>rtos_stack_init(entry, stack_top)</code>	构造初始硬件+软件栈帧，LR 指向 <code>s_thread_exit</code>

中断控制	<code>s_irq_disable(void) / s_irq_enable(level)</code>	使用 PRIMASK 或 BASEPRI
位扫描	<code>__s_ffs(value)</code>	查找最低有效 1 bit (可用 CLZ 或内联循环)
SysTick 钩子	调用 <code>s_tick_increase()</code>	在系统节拍中断中调用

2. Cortex-M 推荐参考实现

本章节可直接复制 Libcpu 下的相应架构文件夹进入工程即可，若不想自己实现可忽略本章节。

2.1 栈初始化

- 硬件自动压栈寄存器顺序（异常返回模式）：
xPSR, PC, LR, R12, R3, R2, R1, R0
- 其余 R4-R11 由 PendSV 软保存
- 设置 xPSR Thumb 位: 0x01000000
- 线程栈必须 8 字节对齐 (Cortex-M EABI)
- LR 指向 `s_thread_exit` (防止线程 return 乱跳)

2.2 上下文切换

典型做法：

1. 在 `s_normal_switch_task` 中存储当前 PSP 到 `*prev`
2. 载入 `*next` 到 PSP
3. 恢复 R4-R11（若采用手动保存方案）
4. 异常返回（BX LR）或直接模拟异常栈返回

也可：

- 设置 PendSV 挂起标志（SCB->ICSR = SCB_ICSR_PENDSVSET_Msk），在 PendSV_Handler 中统一完成保存/恢复。

2.3 s_irq_disable / s_irq_enable

使用 PRIMASK：

```
uint32_t s_irq_disable(void)
{
    uint32_t primask = __get_PRIMASK();
    __disable_irq();
    return primask;
}
void s_irq_enable(uint32_t primask)
{
    if ((primask & 1U) == 0U)
        __enable_irq();
}
```

若需 finer 粒度，可用 BASEPRI。

2.4 __s_ffs 实现

方案一（软件循环）：

```
int __s_ffs(int v)
{
    if (v == 0) return 0;
    int pos = 1;
```

```
while ((v & 1) == 0) { v >>= 1; pos++; }  
return pos;  
}
```

方案二 (CLZ) :

```
int __s_ffs(int v)  
{  
    if (!v) return 0;  
    return 32 - __CLZ(v & -v);  
}
```

3. SysTick 配置

- 令 SysTick 频率 = `START_TICK` (配置

```
SysTick_Config(SystemCoreClock/START_TICK);)
```

- 在 `SysTick_Handler` 中调用 `s_tick_increase()` 函数 :

```
void SysTick_Handler(void)  
{  
    HAL_IncTick();          // 可选  
    s_tick_increase();      // StaRT 内核调度入口  
}
```

4. 中断优先级

- 确保 PendSV (若使用) 优先级最低
- SysTick 优先级高于 PendSV
- 不要在高优先级中断中调用阻塞 API

5. 调试接口重定向

实现 `s_putc` 将输出映射到：

- 串口（阻塞发送）

```
void s_putc(char c)
{
    // 通过串口发送字符
    HAL_UART_Transmit(&huart1, (uint8_t *)&c, 1, HAL_MAX_DELAY);
}
```

- RTT (SEGGER_RTT_PutChar)
- SWO (ITM_SendChar)

6. 验证步骤

1. 初始化：打印横幅
2. 创建两个不同优先级线程：高优先级自增并 yield，低优先级 LED 闪烁
3. 验证 sleep：调用 `s_delay(x)` 是否按 tick 比例调度
4. 验证时间片：相同优先级多个线程轮转
5. 信号量阻塞/释放：Producer/Consumer
6. 线程删除：调用 `s_thread_delete`, idle 清理

7. 堆栈返回：在线程函数末尾不调用 `exit`，验证 `LR->s_thread_exit` 是否安全结束（或主动调用）

7. 常见移植错误

问题	现象	解决
栈未对齐	HardFault	在 <code>rtos_stack_init</code> 处理 8 字节对齐
未正确保存 R4-R11	切换后变量错乱	在汇编保存/恢复低/高寄存器
SysTick 未调用 <code>s_tick_increase</code>	无时间片/睡眠失效	ISR 添调用
<code>__s_ffs</code> 返回错误	调度不工作或死循环	测试位图：输入 0x10 应返回 5
忘记设置 LR	线程 return 后 HardFault	栈帧 LR= <code>s_thread_exit</code>
中断中调用阻塞 API	系统卡死	禁止或判断上下文

完成后系统即可基本运行。