English | **中文**

To RTOS

To RTOS 是一个轻量级实时内核，专为 Cortex-M 设计。目标是使用尽可能少的代码和资源来展示优先级调度、时间片、睡眠定时器和最基本的 IPC 框架。

1. 当前功能

- 固定优先级 + 位图快速查找
- 同优先级时间片轮转（循环）
- 线程生命周期：初始化 / 启动 / 睡眠 / 让出 / 删除 / 重启 / 退出
- 每线程软件定时器（用于睡眠和超时）
- 有序定时器链表
- 轻量级格式化输出 `t_printf`
- IPC 支持：信号量、互斥量（普通和递归）、消息队列
- 平台相关代码独立（汇编上下文切换 + 栈初始化）
- 极低的资源占用：在 -O3 优化下，与基础工程相比，V1.00 仅增加约 1.00 KB FLASH 和 0.17 KB RAM。

2. 目录结构

```
include/        公共内核头文件（例如 ToRTOS.h、tdef.h）
libcpu/CM4F/    Cortex-M4 移植（上下文切换汇编、栈初始化、ffs/fls）
src/            核心模块（调度器、线程、定时器、列表、服务、板级、IPC）
readme/         文档（*.md）
bsp/            板级支持包
```

关键头文件：

- include/ToRTOS.h：公共 API
- include/tdef.h：类型、结构体、宏
- 板级配置：`bsp/.../Core/Inc/ToRTOS_Config.h`（参见 readme/ToRTOS_CONFIG.md）

3. 配置概述

编辑 `ToRTOS_Config.h` 然后全量重建。
基本宏：

- `TO_THREAD_PRIORITY_MAX`
- `TO_TICK`

- `TO_IDLE_STACK_SIZE`
- `TO_USING_IPC`（+ 每个 IPC：SEMAPHORE / MUTEX / RECURSIVE_MUTEX / QUEUE）
- `TO_DEBUG`
  详情请见：readme/ToRTOS_CONFIG.md

4. 快速上手（伪流程）

```
#include "ToRTOS.h"

#define STACK_SZ 512
static t_thread_t t1;
static t_uint8_t t1_stack[STACK_SZ];

static void thread1_entry(void *arg)
{
    while (1)
    {
        t_printf("t1 running, tick=%d\n", (int)t_tick_get());
        t_mdelay(1000);
    }
}

int main(void)
{
    /* 硬件初始化（时钟、UART putc 重写、SysTick -> t_tick_increase）*/

    t_tortos_init();                  /* 核心初始化：调度器、定时器、空闲、横幅 */

    t_thread_create_static(thread1_entry, t1_stack, STACK_SZ, 5, NULL, 10, &t1);
    t_thread_startup(&t1);

    t_sched_start();                 /* 永不返回 */
    while (1);
}
```

SysTick 处理程序必须调用：

```
void SysTick_Handler(void)
{
    t_tick_increase();
}
```

重写字符输出（示例）：

```
void t_putc(char c)
{
    /* UART 发送或 ITM_SendChar(c); */
}
```

5. 核心 API（快照）

来自 include/ToRTOS.h：

- 线程：`t_thread_create_static`、`t_thread_create`（动态）、`t_thread_startup`、`t_thread_sleep`、`t_thread_exit`、`t_thread_delete`、`t_thread_restart`

- 调度器控制：`t_sched_start`
- 定时：`t_mdelay`、`t_tick_get`
- IPC：信号量（`T_SEMA_*`）、互斥量（`T_MUTEX_*`）、递归互斥量（`T_MUTEX_RECURSIVE_*`）、队列（`T_QUEUE_*`）
- 调试 / 日志：`t_printf`、`T_DEBUG_LOG`

返回代码：`t_status_t`（参见 include/tdef.h）

6. 时间模型

- 全局 tick：在 SysTick 中通过 `t_tick_increase()` 递增
- 每线程时间片重载：`init_tick`
- 睡眠：每线程定时器插入有序列表；到期回调使线程就绪
- 有符号时间比较处理回绕

7. 移植（总结）

更多移植相关（参见扩展指南 readme/ToRTOS_TRANS.md）：

- 上下文切换汇编：`t_first_switch_task`、`t_normal_switch_task`、PendSV 处理程序
- 栈帧布局在 `t_stack_init`
- IRQ 屏蔽：`t_irq_disable / t_irq_enable`
- Tick 源调用 `t_tick_increase`
- 可选 `__t_ffs` / `__t_fls`（可回退到内置或循环）

8. 编码风格原则

- 所有队列使用侵入式循环双向链表
- 关键段：仅 IRQ 禁用（短）
- 默认静态分配，可选动态分配
- 最小内联汇编隔离

9. 许可证

10. 贡献

1. Fork / 分支
2. 保持模块小而隔离
3. 添加简要 Doxygen 注释
4. 提交带测试描述的 PR

11. 最小故障排除

症状	可能原因	补救措施
无上下文切换	SysTick 缺失或 `__t_ffs` / `__t_fls` 错误	检查处理程序 + 位图
睡眠永不唤醒	`t_tick_increase` 未调用	验证 SysTick_Handler
输出乱码	并发 `t_printf`	接受或用锁包装
启动时 HardFault	栈未 8 字节对齐	检查 `t_stack_init`

祝使用 ToRTOS 愉快!