English | 中文

StaRT RTOS

A small educational real- time kernel for Cortex- M targets. Focus: clarity of core mechanisms (priority bitmap scheduler, tick timer, basic thread lifecycle, simple IPC placeholders).

1. Features (Current)

- Preemptive fixed- priority scheduler (0 = highest) with bitmap fast lookup
- Round- robin within same priority via time slice
- Thread lifecycle: init / startup / sleep / yield / delete / restart / exit
- Per- thread software timer (used for sleep & timeouts)
- Simple ordered timer list (O(n) insert for now)
- Lightweight formatted output `s_printf` (supports %d %s %c)
- Optional IPC scaffolding (semaphore / mutex / message queue stubs — semaphore partly implemented)

- Clean separation of arch port (context switch, stack frame, irq mask)

Planned: full IPC implementations, priority inheritance, message queue, tickless, stack watermark, diagnostics.

2. Directory Layout

```
include/         Public kernel headers (e.g. start.h, sdef.h)
libcpu/CM3/      Cortex-M3 port (context switch asm, stack init, ffs)
src/             Core modules (scheduler, thread, timer, list, service,
board, ipc draft)
readme/          Documentation (*.md)
bsp/             Board support packet
```

Key headers:

- include/start.h: Public API
- include/sdef.h: Types, structs, macros
- Board config: `bsp/.../Core/Inc/StaRT_Config.h` (see readme/StaRT_CONFIG.md)

3. Configuration Overview

Edit `StaRT_Config.h` then full rebuild.
Essential macros:

- `START_THREAD_PRIORITY_MAX`
- `START_TICK`
- `START_IDLE_STACK_SIZE`
- `START_USING_IPC` (+ per IPC: SEMAPHORE / MUTEX / MESSAGEQUEUE)

- `START_DEBUG`

  See details: StaRT_CONFIG.md

4. Quick Start (Pseudo Flow)

c

```c
#include "start.h"

#define STACK_SZ 512
static s_thread t1;
static s_uint8_t t1_stack[STACK_SZ];

static void thread1_entry(void)
{
    while (1)
    {
        s_printf("t1 running\n");
        s_mdelay(1000);
    }
}

int main(void)
{
    /* Hardware init (clock, UART putc override, SysTick ->
s_tick_increase) */

    s_start_init();                  /* core init: scheduler, timers,
idle, banner */

    s_thread_init(&t1, thread1_entry, t1_stack, STACK_SZ, 5, 10);
    s_thread_startup(&t1);

    s_sched_start();                 /* never returns */
    while (1);
}
```

SysTick handler must call:

c

```c
void SysTick_Handler(void)
{
    s_tick_increase();
}
```

Override character output (example):

c

```
void s_putc(char c)
{
    /* UART transmit or ITM_SendChar(c); */
}
```

5. Core APIs (Snapshot)

From include/start.h:

- Thread: `s_thread_init`, `s_thread_startup`, `s_thread_sleep`, `s_thread_yield`, `s_thread_exit`, `s_thread_delete`, `s_thread_restart`
- Scheduler control: `s_sched_start`
- Timing: `s_mdelay`, `s_tick_get`
- Semaphore (partial): `s_sem_init`, `s_sem_take`, `s_sem_release`
- Debug / log: `s_printf`, `S_DEBUG_LOG`

Return codes: `s_status` (see include/sdef.h)

6. Timing Model

- Global tick: incremented in SysTick via `s_tick_increase()`
- Time slice reload per thread: `init_tick`
- Sleep: per-thread timer inserted in ordered list; expiration callback readies thread
- Signed time comparisons handle wraparound

7. Porting (Summary)

More related to transplantation (see extended guide StaRT_TRANS.md):

- Context switch assembly: `s_first_switch_task`, `s_normal_switch_task`, PendSV handler
- Stack frame layout in `s_stack_init`
- IRQ mask: `s_irq_disable / s_irq_enable`
- Tick source calling `s_tick_increase`
- Optional `__s_ffs` (can fall back to builtin or loop)

8. Coding Style Principles

- Intrusive circular doubly linked lists for all queues
- Critical sections: IRQ disable only (short)
- No dynamic allocation in core (user supplies thread stacks statically)
- Minimal inline assembly isolation

9. Limitations / Known Gaps

- IPC (mutex, message queue) not fully implemented
- No memory manager / heap
- No priority inheritance yet
- No stack overflow detection
- Timer list O(n); no skiplist levels >1

• Logging not thread-safe (acceptable for demo)

10. License

```
Copyright (c) 2025 StaRT
SPDX-License-Identifier: MIT
```

11. Contributing

1. Fork / branch
2. Keep modules small & isolated
3. Add brief Doxygen comments
4. Submit PR with test description

12. Minimal Troubleshooting

Symptom	Likely Cause	Remedy
No context switch	SysTick missing or `__s_ffs` wrong	Check handler + bitmap
Sleep never wakes	`s_tick_increase` not invoked	Verify SysTick_Handler
Output garbled	Concurrent `s_printf`	Accept or wrap with lock

HardFault on start	Stack not 8-byte aligned	Inspect `s_stack_init`

Happy hacking with StaRT!