Software Design Description (SDD)

Progetto: RTOS minimale in Rust per RISC-V (M-mode)
Data: 2025-08-31

---

1. Panoramica

Un piccolo RTOS cooperativo che fornisce: tick di sistema, gestione task, primitive di sincronizzazione (semaforo/spinlock), console UART e gestione trap/panic. Il codice assembly fornisce l'entry point delle trap e il bootstrap del primo task.

---

2. Architettura

2.1 Struttura dei Moduli

- **arch/**
  - `mod.rs` — definizioni delle costanti del trap frame e dichiarazioni `extern` (`trap_entry`, `__rtos_boot_with_sp`, simboli per gli stack dei task).
  - `trap.rs` — `trap_handler` legge `mcause`/`mepc` e smista verso ISR del timer o i fault handler.
  - `timer.rs` — `init_timer(ticks)`, `timer_interrupt()`.
- **kernel/**
  - `task.rs` — Task Control Block, creazione task, stati ready/run, avvio primo task, funzioni di scheduling.
  - `services.rs` — contatore tick, `delay_ms`, `task_yield`, `Semaphore`, `SpinLock`.
- **drivers/**
  - `uart.rs` — trasmissione UART in polling (`puts`, `put_hex`, `put_dec`).
- **panic_handler.rs** — routine di panic con disabilitazione interrupt e diagnostica.
- **main.rs** — workload dimostrativo, stampe UART, inizializzazione timer, creazione task e avvio.

2.2 Flusso di Controllo

1. **Boot**: startup in assembly -> Rust `main` (o `entry`) -> `init_timer(+ticks)` -> creazione task -> `start_first_task()` -> `__rtos_boot_with_sp`.
2. **Tick**: interrupt del timer -> `timer_interrupt()` -> `rtos_on_timer_tick()` (incrementa tick, sblocca delay) -> ritorno al contesto preempted (scheduling cooperativo).
3. **Yield/Block**: i task chiamano `task_yield()` o `delay_ms()`/`Semaphore::wait()`, passando allo stato **Blocked/Ready**.
4. **Trap/Fault**: `trap_handler` distingue timer vs eccezioni; i fault portano a panic.

2.3 Struttura dei Dati

- **TCB (`task.rs`)**
  Campi: `sp: *mut usize`, `entry: extern "C" fn()`, `state: TaskState`, `priority: u8`, `stack_lo/hi`.
  Limiti: `MAX_TASKS = 4`, `TASK_STACK_BYTES = 4096`.
- **Timebase Globale (`services.rs`)**
  `TICKS: AtomicU64`; `TICK_HZ = 1000` (1 ms).
- **Primitive di Sincronizzazione**
  - `SpinLock<T>` con mutabilità interna per sezioni critiche brevi.
  - `Semaphore` con contatore atomico e attesa bloccante.

2.4 Interfacce

- **Funzioni Pubbliche**
  - Tempo: `ticks()`, `delay_ms(ms: u64)`, `task_yield()`
  - Task: `create_task(entry, prio) -> handle`, `start_first_task()`
  - Sync: `Semaphore::new(n)`, `wait()`, `post()`, `try_wait()`; `SpinLock::new(v)`, `lock()`
  - UART: `puts(&str)`, `put_hex(usize)`, `put_dec(usize)`
- **ISR/Trap**
  - `timer_interrupt()` invocata da `trap_handler` su causa timer.
  - Il percorso di panic stampa causa, `mepc`, `mtval` e ferma il sistema.

2.5 Temporizzazione

- Periodo tick: 1 ms (QEMU `virt` MTIME =10 MHz, riarmo +10_000).
- WCET ISR: O(1) (riarmo + sblocco), nessuna allocazione dinamica.

2.6 Memoria

- Stack per task: 4096 B (configurabile).
- Stack dei task allocati nella sezione `.tasks` definita dal linker (`__task_stack_start/end`).
- Dimensione trap frame: `FRAME_WORDS = 20` (registri salvati in assembly) (19 usati, 1 di padding).

2.7 Gestione Errori

- Istruzione illegale o fault memoria -> panic.
- Panic: interrupt disabilitati, messaggio emesso, loop infinito.

2.8 Criteri e Motivazioni Progettuali

- Lo scheduling cooperativo semplifica l'analisi di determinismo e safety.
- UART in polling riduce la complessità del driver.
- Uso di atomiche per tick e semafori, evitando di disabilitare interrupt per sezioni lunghe.

---

3. Matrice di Tracciabilità

Req ID	Design/Code Reference	Verifica (SVCP)
HLR-1	`timer.rs::init_timer`, `timer_interrupt`; `services.rs::rtos_on_timer_tick`	TBD
HLR-2	`task.rs` (TCB, scheduler helpers)	TBD
HLR-3	`task.rs::create_task`, `task.rs::start_first_task`, `services.rs::task_yield`, `mod.rs::__rtos_boot_with_sp`	TBD
HLR-4	`services.rs::delay_ms`	TBD
HLR-5	`services.rs::Semaphore`	TBD
HLR-6	`services.rs::SpinLock`	TBD
HLR-7	`uart.rs::puts`, `put_hex`, `put_dec`	TBD
HLR-8	`trap.rs::trap_handler`, `panic_handler.rs::panic_handler`	TBD
HLR-9	`timer.rs::init_timer`, `task.rs::start_first_task`	TBD
HLR-10	`mod.rs::trap_entry`, `__rtos_boot_with_sp`	TBD
LLR-1	`timer.rs::init_timer` (+10_000)	TBD
LLR-2	`services.rs::ticks`	TBD
LLR-3	`services.rs::delay_ms`	TBD
LLR-4	`task.rs::MAX_TASKS`, `TASK_STACK_BYTES`	TBD
LLR-5	`task.rs::TCB` (campi: SP, entry, state, priority, stack)	TBD

Req ID	Design/Code Reference	Verifica (SVCP)
LLR-6	`task.rs::start_first_task`, `mod.rs::__rtos_boot_with_sp`	TBD
LLR-7	`services.rs::task_yield`	TBD
LLR-8	`services.rs::Semaphore::{wait,post,try_wait}`	TBD
LLR-9	`services.rs::SpinLock`	TBD
LLR-10	`uart.rs::{puts, put_hex, put_dec}`	TBD
LLR-11	`trap.rs::trap_handler` (mcause, mepc decode)	TBD
LLR-12	`timer.rs::timer_interrupt`	TBD
LLR-13	`panic_handler.rs::panic_handler`	TBD
LLR-14	`mod.rs::OFF_*` costanti + `trap.S` frame	TBD
NFR-1	determinismo O(1) in servizi (`services.rs`, `task.rs`)	TBD
NFR-2	allocazioni statiche (`task.rs`, `extra-sections.x`)	TBD
NFR-3	footprint `.tasks` 16 KiB (`extra-sections.x`)	TBD
NFR-4	gestione errori → panic (`panic_handler.rs`, `trap.rs`)	TBD

4. Assunzioni

- **DAL**: livello da assegnare (per questo progetto non viene attribuito, contesto dimostrativo/accademico).
- **Context switch assembly** (`trap_entry`, salvataggio/ripristino registri) fornito separatamente e coerente con i valori `OFF_*`. Verificato: 19 registri salvati, 1 slot extra riservato (padding/allineamento).