Software Requirements Data (SRD)

Progetto: RTOS minimale in Rust per RISC-V (M-mode)
Standard di riferimento: DO-178C (obiettivi applicabili a un RTOS minimale) **Data:** 2025-08-31

---

1. Scopo

Il presente documento definisce i requisiti software per un sistema operativo real-time minimale (RTOS) sviluppato in Rust per architettura RISC-V (RV64) in modalità *Machine* (M-mode), eseguito su macchina virtuale QEMU (`virt`).
L'RTOS fornisce un piccolo kernel con scheduling cooperativo (`task_yield`), tick di sistema a 1 ms, primitive di sincronizzazione di base e I/O su UART per debug e/o logging.
L'obiettivo è descrivere requisiti **chiari, verificabili e tracciabili**, in linea con i principi della DO-178C.

Fuori dal perimetro: protezione memoria (MMU/MPU), file system, stack di rete, modalità *User*. I moduli assembly di bootstrap e di cambio contesto sono trattati come interfacce esterne.

---

2. Riferimenti

- Codice sorgente:
  - `main.rs`,
  - `panic_handler.rs`,
  - `task.rs`,
  - `services.rs`,
  - `uart.rs`,
  - `trap.rs`,
  - `timer.rs`,
  - `mod.rs`
- Hardware: QEMU `virt`; con CLINT/MTIME e UART0 (MMIO).
- Standard: DO-178C (RTCA DO-178C / EUROCAE ED-12C).

---

3. Ambiente Operativo e Vincoli

- **CPU**: RISC-V RV64 in modalità Machine.
- **Timer**: CLINT MTIME/MTIMECMP =10 MHz (QEMU `virt`).
- **UART0**: base `0x1000_0000`, TX a offset 0 e LSR a offset 5.
- **Interrupt**: utilizzati esclusivamente gli interrupt di Machine Timer; gli interrupt esterni e software non sono gestiti.
- **Build**: il sistema è sviluppato in Rust, compilato per il target `riscv64imac-unknown-none-elf` ed eseguito senza libreria standard (`no_std`).

---

4. Definizioni

- **Tick**: unità temporale del kernel (1 ms).
- **Task**: funzione eseguita dal kernel con stack e TCB dedicati.
- **TCB**: Task Control Block che contiene SP, entry, stato, priorità, limiti stack.
- **Bloccante**: stato di attesa di un evento (semaforo o delay).

---

5. Requisiti Funzionali di Alto Livello (HLR)

- **HLR-1 — Clock/Tick**: Il kernel deve mantenere un contatore di sistema che si incrementa regolarmente ogni 1 ms tramite interrupt di Machine Timer.
- **HLR-2 — Scheduler**: Il kernel deve gestire fino a 4 task concorrenti, eseguendone uno per volta.
- **HLR-3 — API Task**: Il kernel deve fornire API per creare task, cedere la CPU (*yield*) e avviare il primo task.
- **HLR-4 — Delay**: Il kernel deve fornire un servizio di attesa temporizzata in millisecondi.
- **HLR-5 — Semafori**: Il kernel deve fornire semafori contatori (`post`, `wait`, `try_wait`).
- **HLR-6 — Spinlock**: Il kernel deve fornire spinlock per brevi sezioni critiche.
- **HLR-7 — Console I/O**: Il kernel deve fornire funzioni UART per stringhe e valori numerici.
- **HLR-8 — Trap/Panic Handling**: Il kernel deve distinguere eccezioni/interrupt e fornire un gestore di panic che disabilita gli interrupt, stampa cause/PC e ferma il sistema. Gli interrupt esterni sono considerati fuori dal perimetro.
- **HLR-9 — Avvio Deterministico**: Il kernel deve inizializzare timer e primo task in maniera deterministica.
- **HLR-10 — Interfacce Assembly**: Il kernel deve esporre interfacce architetturali (`trap_entry`, `__rtos_boot_with_sp`) implementate in assembly.

---

6. Requisiti Funzionali di Basso Livello (LLR)

- **LLR-1 — Frequenza Tick**: Il kernel deve riarmare `MTIMECMP` di +10.000 cicli (QEMU `virt`, 10 MHz) generando un tick da 1 ms. (`timer.rs`)
- **LLR-2 — Contatore Tick**: `ticks()` deve restituire un contatore a 64 bit. (`services.rs`)
- **LLR-3 — Conversione Delay**: `delay_ms(ms)` deve bloccare finché `ticks() >= t0 + ms`. (`services.rs`)
- **LLR-4 — Limiti Task**: `MAX_TASKS = 4`; `TASK_STACK_BYTES = 4096`. (`task.rs`)
- **LLR-5 — Contenuto TCB**: Ogni TCB deve includere SP, entry, stato, priorità, limiti stack. (`task.rs`)
- **LLR-6 — Avvio Primo Task**: `start_first_task()` deve avviare il primo Ready task tramite `__rtos_boot_with_sp`. (`task.rs`, `mod.rs`)
- **LLR-7 — Yield**: `task_yield()` deve passare la CPU al prossimo Ready di pari priorità. (`services.rs`, `task.rs`)
- **LLR-8 — Correttezza Semaforo**: `wait()` blocca se `count==0`; `post()` incrementa; `try_wait()` decrementa se count>0. (`services.rs`)
- **LLR-9 — SpinLock**: `lock()` attende attivamente, `release()` su drop. (`services.rs`)
- **LLR-10 — UART TX**: `puts`, `put_hex`, `put_dec` devono scrivere via polling. (`uart.rs`)
- **LLR-11 — Decodifica Trap**: `trap_handler` deve leggere `mcause` e `mepc`, gestendo timer ed eccezioni. (`trap.rs`)
- **LLR-12 — ISR Timer**: `timer_interrupt()` deve chiamare `rtos_on_timer_tick()` e riarmare `MTIMECMP`. (`timer.rs`)

- **LLR-13 — Azioni Panic**: in panic il sistema deve disabilitare interrupt, stampare diagnostica (cause/PC) e fermarsi. (`panic_handler.rs`)
- **LLR-14 — Layout Frame**: il layout (`OFF_*`) deve corrispondere al salvataggio assembly. (`mod.rs`)

---

7. Requisiti Non Funzionali (NFR)

- **NFR-1 — Determinismo**: Tutti i servizi del kernel devono essere deterministici; le primitive base (`yield`, `delay`, `Semaphore`, `SpinLock`) devono avere complessità O(1).
- **NFR-2 — Allocazione Memoria**: Non deve essere usata allocazione dinamica; tutte le strutture (TCB, stack) devono essere allocate staticamente.
- **NFR-3 — Footprint**: Lo spazio totale riservato agli stack dei task deve essere di 16 KiB (4 task × 4096 B), allocato nella sezione `.tasks`.
- **NFR-4 — Sicurezza in errore**: In caso di eccezioni non previste, il kernel deve entrare in modalità di *panic* e fermarsi in modo sicuro.

---

8. Interfacce

- **Servizi pubblici**: `ticks`, `delay_ms`, `task_yield`, `Semaphore`, `SpinLock`, UART print.
- **Architettura**: `trap_entry`, `__rtos_boot_with_sp`, ISR timer.
- **Configurazione**: `TICK_HZ`, `MAX_TASKS`, `TASK_STACK_BYTES`.

---

9. Considerazioni di Sicurezza e Assurance

- Livello di garanzia (DAL): non assegnato (progetto accademico/dimostrativo).
- Misure difensive: panic disattiva interrupt e segnala stato; fault non gestiti portano a panic.
- WCET/latency: spinlock brevi; ISR timer O(1). Delay e semafori sono bloccanti.

---

10. Verifica

Ogni requisito HLR/LLR/NFR è verificato tramite casi di test nel documento **SVCP**. Criteri di successo/fallimento osservabili via UART o strumenti di analisi.

---

11. 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

Req ID	Design/Code Reference	Verifica (SVCP)
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
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

Req ID	Design/Code Reference	Verifica (SVCP)
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