

Scheduler facade i scoped guards

statyczna usługa; RAII tylko dla prawdziwych par

Karta	K09 / 16	Czas	30 minut
Platforma	Hazard3 / RV32I	Język	freestanding C++17
Dowód	nesting + mstatus.MIE	API	task context
Wersja	v00.01	UUID	6a2ab068-...
		karty	

Najpierw klasyfikacja lifetime

`Scheduler::start/state` → usługa procesu, brak scoped lifetime
`enter/exit, suspend/resume` → pary, więc guard RAII

Kontrakt K09. Destruktor guarda ma odtworzyć stan również przy każdym `return`. Nie wolno jednak blokować przy zawieszonym schedulerze, a `CriticalSection` nie jest mutexem ani API dla ISR.

Plan 30 minut

Czas	Tryb	Dowód
0–5	raw API	rozdziel usługi od par
5–10	critical	nesting oraz MIE przed/wewnątrz/po
10–15	guard	trzy wczesne wyjścia wracają do stanu bazowego
15–20	suspend	scheduler suspended, lecz MIE pozostaje 1
20–25	nesting	inner exit nie wznowia jeszcze outer suspension
25–30	ABI	inline do raw API; brak vtable/runtime

Dwie różne blokady

Operacja	Context switch	Przerwania	Para
<code>vTaskSuspendAll</code>	odroczony	działają	<code>xTaskResumeAll</code>
<code>taskENTER_CRITICAL</code>	niemożliwy	MIE=0	<code>taskEXIT_CRITICAL</code>

 Predykcja MIE podczas suspension: _____. Predykcja critical depth po dwóch enter i jednym exit: _____.

1 Raw critical: zmierz dokładną sekwencję

```
taskENTER_CRITICAL(); // depth + 1, MIE=0
taskENTER_CRITICAL(); // depth + 1, MIE=0
taskEXIT_CRITICAL(); // depth - 1, nadal MIE=0
taskEXIT_CRITICAL(); // depth == 0, MIE=1
```

Checkpoint	Operacja	Predykcja depth/MIE	Odczyt depth/MIE
C0	przed	0 / 1	_____
C1	enter	1 / 0	_____
C2	drugi enter	2 / 0	_____
C3	pierwszy exit	1 / 0	_____
C4	drugi exit	0 / 1	_____

2 RAII zamienia trzy krawędzie w jedną parę

```
uint32_t repair(uint32_t path) {
    freertos::CriticalSection guard;
    if (path == 0) return 0x10;
    if (path == 1) return 0x20;
    return 0x30;
} // destructor runs on every return
```

Path	Wynik	inside depth/MIE	after depth/MIE	PASS
0	0x10	_____	_____	☐
1	0x20	_____	_____	☐
2	0x30	_____	_____	☐

Co dokładnie gwarantuje guard?

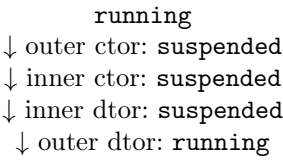
- paruje wywołania w normalnym przepływie i przy wcześniejszym return;
 - nie jest kopiowalny ani przenoszalny, więc nie powiela exit;
 - zachowuje zagnieżdżenie: inner exit nie kończy outer critical section.
- Czy RAII pozwala wywołać blokujące API wewnątrz critical section? _____. Uzasadnienie:
-

3 SchedulerSuspendGuard: context switch bez maskowania IRQ

```
{
    freertos::SchedulerSuspendGuard guard;
    // scheduler state == suspended; mstatus.MIE == 1
    // no delay, queue wait, semaphore wait or other blocking API
} // xTaskResumeAll exactly once
```

Path	Inside state/MIE	After state/MIE	Wynik
0	suspended / _____	running / _____	0x40
1	suspended / _____	running / _____	0x50
2	suspended / _____	running / _____	0x60

Zagnieżdżenie



Dlaczego inner destructor nie może jeszcze zwrócić stanu running?

4 Statyczna fasada i bieżący task

```
freertos::Scheduler::start();
auto state = freertos::Scheduler::state();
auto handle = freertos::this_task::current();
auto tick = freertos::this_task::tick_count();
freertos::this_task::yield();
```

Relacja	Predykcja	Odczyt
state przed start	not started	_____
state w workerze	running	_____
this task handle	worker handle	_____
tick after yield	≥ tick before	_____

Granice

Suspension nie wyłącza ISR; tylko odracza przełączenie taska. Critical section maskuje przerwania i musi być krótka. Żaden z guardów nie jest zamiennikiem mutexa chroniącego zasób przez długą operację.

5 Hazard3/GDB i ABI

```
make check
riscv64-unknown-elf-gdb build/task01_kernel_facade/prog.elf
b kernel_facade_debug_checkpoint
```

```
p g_raw_critical_depth
p g_raw_critical_mie
p g_guard_critical_after_depth
p g_guard_critical_after_mie
p g_raw_scheduler_state
p g_guard_scheduler_after_state
p g_nested_scheduler_state
p g_this_task_handle
p g_worker_handle
p g_kernel_facade_pass
```

Co ma zostać w assembly?

Dowód	Oczekiwane	Odczyt
disable IRQ	csrci mstatus,8	_____
enable IRQ	csrsi mstatus,8	_____
symbole wrappera	brak	_____
vtable/RTTI/exception runtime	brak	_____

Zaliczenie

- ☐ rozdzielałam statyczną fasadę od scoped ownera;
- ☐ odtwarzam critical depth 0–1–2–1–0 i MIE 1–0–0–0–1;
- ☐ pokazuję restore po trzech wczesnych return;
- ☐ pokazuję, że suspension zachowuje MIE=1;
- ☐ wyjaśniam nesting outer/inner i zakaz blokowania;
- ☐ dowodzę inline facade/guards oraz końcowy PASS.

Wyjście

Kiedy wybrać `CriticalSection`, a kiedy `SchedulerSuspendGuard`?

Dlaczego `Scheduler` nie jest obiektem RAII?

Następna karta K10: `typed Queue<T>` i `StaticQueue<T,N>` z jawnym kontraktem copyable message type.