

Scheduler: stany, priorytety i typowane ticki

przewidź przełączenie, potem udowodnij je śladem

Karta	K04 / 16	Czas	30 minut
Platforma	Hazard3 / RV32I	Język	freestanding C++17
Scheduler	preemption + time slicing	Kernel	V11.3.0, bez zmian
Wersja	v00.01	UUID	853a7dd7-... .
		karty	

Cel i jedna sekwencja dowodowa

Dwa zadania A i B mają priorytet 2. Najpierw scheduler dzieli CPU między równych. Potem A blokuje się na 3 ticki. Na końcu A podnosi priorytet B do 3. Nie zgadujemy kolejności — zapisujemy stany, ticki oraz znaczniki wykonania.

READY → RUNNING → BLOCKED → READY → RUNNING → DELETED

Dwa warunki K04. Zadanie o najwyższym priorytecie wygrywa tylko wtedy, gdy jest gotowe. Tick jest jednostką schedulera, nie ukrytą nazwą milisekundy.

Plan 30 minut

Czas	Tryb	Dowód ucznia
0–4	predykcja	tabela stanów i bieżącego zadania
4–8	czas typowany	Ticks kontra TickPoint
8–13	time slicing	A–B–A albo B–A–B przy priorytecie 2
13–19	blokada	A=BLOCKED, B=RUNNING, różnica ticków 3
19–25	priorytet	BEFORE–HIGH–AFTER ma wartości 1–2–3
25–30	wyjście	delay względny/okresowy i granica TaskRef

Najpierw przewidź

Który identyfikator pojawi się pierwszy: A czy B? _____ Czy test powinien wymagać właśnie tej odpowiedzi?

1 Stany są warunkiem kwalifikacji do CPU

Stan	Czy może teraz dostać CPU?	Typowa przyczyna
eRunning	tak, jedno zadanie na rdzeń	scheduler właśnie je wybrał
eReady	tak, czeka na wybór	równy/niższy priorytet lub wywłaszczenie
eBlocked	nie	delay albo oczekiwanie na zdarzenie
eSuspended	nie	jawne zawieszenie, nie upływ czasu
eDeleted	nie	stan terminalny w śladzie tej karty

Pułapka „najwyższy zawsze działa”

Jeżeli A ma priorytet 4, ale jest BLOCKED, a B ma priorytet 2 i jest READY, scheduler wybierze _____, ponieważ _____.

Równe priorytety i time slicing

Przy configUSE_PREEMPTION=1 oraz configUSE_TIME_SLICING=1 tick może przenieść wykonanie do innego gotowego zadania o tym samym priorytecie. Zalicza relacja:

```
first == third && first != second
```

Wpisz dwa dopuszczalne ślady: _____ lub _____.

2 Czas: wartość ma znaczenie dopiero z jednostką

```
freertos::Ticks duration{3};
const auto start = freertos::TickPoint::now();
vTaskDelay(duration.count());
const auto elapsed = freertos::TickPoint::now() - start;
```

Ticks jest czasem trwania. TickPoint jest punktem na zawijającym się liczniku. Odejmowanie dwóch punktów daje czas trwania także przy przejściu licznika przez maksimum typu bez znaku.

API	Znaczenie
vTaskDelay(3)	blokada względna: trzy ticki od chwili wywołania
xTaskDelayUntil(&next, 3)	kolejny termin względem zapamiętanej osi okresu

W laboratorium tick rate wynosi 1000 Hz, ale poprawne zdanie brzmi „A było zablokowane przez _____ ticki”. Przeliczenie na ms: _____. Co zmieni się przy 250 Hz? _____

3 Tabela predykcji i obserwacji

Najpierw wypełnij kolumnę „predykcja”. Dopiero potem uruchom program i wpisz wartości z `g_snapshots`. Priorytet zapisuj obok stanu, np. `READY/p2`.

Nr	Zdarzenie	A: pred./odczyt	B: pred./odczyt	Current	Dlaczego?
1	startup	_____	_____	_____	_____
2	przed delay	_____	_____	_____	_____
3	A czeka	_____	_____	_____	_____
4	A obudzone	_____	_____	_____	_____
5	BEFORE	_____	_____	_____	_____
6	HIGH	_____	_____	_____	_____
7	AFTER	_____	_____	_____	_____
8	verifier	_____	_____	_____	_____

Trzy znaczniki jako dowód wywłaszczenia

Kod A zapisuje `BEFORE=1`, podnosi B do priorytetu 3, a po powrocie z API zapisuje `AFTER=3`. Kod B zapisuje `HIGH=2` natychmiast po uzyskaniu CPU.

```
append_priority_order(BEFORE);      // A: 1
peer.set_priority(3);                // B staje sie wyzej priorytetowe
append_priority_order(AFTER);        // A: 3, dopiero po powrocie

// B, zanim wywołanie w A zdazy wrocic:
append_priority_order(HIGH);         // B: 2
```

Gdyby nie doszło do natychmiastowego wywłaszczenia, możliwa kolejność wynosiłaby _____.
Odczyt oczekiwany: _____.

Osiem rekordów, ale żadnego użycia wiszącego uchwytu

Po usunięciu zadania jego TCB może zostać odzyskany przez idle task. `TaskRef` nie posiada TCB i nie wydłuża jego życia. Dlatego terminalny rekord zapisuje znany stan `DELETED`, ale nie wywołuje `eTaskGetState()` przez stary uchwyt.

4 Uruchomienie i kontrakt dowodu

```
make check
riscv64-unknown-elf-gdb build/task01_scheduler_states/prog.elf
```

```
p g_alternating_tasks
p g_alternating_ticks
p g_snapshots
p g_delay_start_tick
p g_delay_observed_tick
p g_delay_end_tick
p g_delay_elapsed
p g_priority_order
p g_scheduler_states_pass
```

Warunek	Odczyt ucznia
identyfikatory mają postać X,Y,X, $X \neq Y$	_____
w zdarzeniu 3: A=BLOCKED, B=RUNNING	_____
<code>delay_end - delay_start == 3</code>	_____
obserwacja blokady jest przed końcem delay	_____
w zdarzeniu 6 B ma priorytet 3	_____
<code>priority_order == {1,2,3}</code>	_____
osiem snapshotów i <code>pass == 1</code>	_____

Minimalne wrappery C++

```
class TaskRef final {
public:
    explicit TaskRef(TaskHandle_t h) : handle_{h} {}
    eTaskState state() const { return eTaskGetState(handle_); }
    UBaseType_t priority() const { return uxTaskPriorityGet(handle_); }
    void set_priority(UBaseType_t p) const {
        vTaskPrioritySet(handle_, p);
    }
private:
    TaskHandle_t handle_; // non-owning
};
```

Nie ma tu wirtualności, alokacji ani automatycznego delete. Rozmiar `TaskRef` jest równy rozmiarowi uchwytu. Pełna polityka własności taska pojawia się dopiero w K07–K09.

Wyjście — cztery krótkie odpowiedzi

1. Dlaczego zadanie o priorytecie 4 może nie działać, gdy działa p2?
2. Dlaczego test nie wymaga, by A było pierwsze w śladzie time slicing?
3. Kiedy `vTaskDelay(3)` oznacza dokładnie 3 ms?
4. Co dokładnie dowodzi kolejność 1,2,3?

Zaliczenie

- ☐ rozróżniam READY, RUNNING, BLOCKED i DELETED;
- ☐ pokazuję X–Y–X, blokadę na 3 ticki i porządek 1–2–3;
- ☐ odróżniam delay względny od okresowego terminu;
- ☐ nie utożsamiam ticka z milisekundą bez konfiguracji;
- ☐ wiem, że `TaskRef` nie jest właścicielem TCB.

Następna karta K05: most C++ do heapu, porównanie modeli FreeRTOS oraz tylko-odczytowy `HeapStats`.