

FreeRTOS heap_4: od kursora do wolnych bloków

bezpośrednia kontynuacja projektu K&R 5.4

Karta	FC01 / 15	Czas	30 minut
Płytką	Raspberry Pi Pico 2 W	Układ	RP2350, rdzeń RISC-V
Allocator	FreeRTOS-Kernel V11.3.0	Źródło	upstream heap_4.c
Wersja	v0.3	Data	19.07.2026

Punkt startowy — tego już nie powtarzamy

Z poprzedniej karty działa projekt:

```
allocbuf[64];  allocp = allocbuf;
alloc_local(5) -> offset 0
alloc_local(7) -> offset 5
allocp        -> offset 12
alloc_local(0) i OOM -> NULL; kursor pozostaje na 12
```

Pytanie projektowe. Jak zwolnić obszar 5 B spod offsetu 0, zachować późniejszy obszar 7 B spod offsetu 5 i ponownie wykorzystać powstałą dziurę? W naszym projekcie nie istnieje `free`; reset odzyskuje całą arenę. Samo cofnięcie kursora zniszczyłoby również późniejsze przydziały.

Cel lekcji

Potrafię wskazać trzy elementy dodane przez `heap_4` do alokatora liniowego — **nagłówek bloku**, **listę wolnych bloków** i **scalanie** — oraz potwierdzić ich działanie na prawdziwym `heap_4.c`.

Plan 30 minut

Czas	Tryb	Wynik
0–3	problem K&R	nazwać, dlaczego jeden kursor nie zwolni obszaru ze środka
3–4	kontekst	FreeRTOS linkuje jedną implementację; tutaj pracujemy tylko z <code>heap_4</code>
4–9	Demo 1	przewidzieć nagłówek, first-fit i split
9–16	Demo 2	przewidzieć listę adresową i scalenie 2 -> 1
16–27	Task 3 RUN	dwa checkpointy prawdziwego <code>heap_4.c</code> na Pico 2 W
27–30	wyjście	odróżnić total free, largest free i minimum-ever

Jedna minuta kontekstu: `heap_1` przypomina rosnący kursor bez `free`; `heap_2` nie scala; `heap_3` używa alokatora biblioteki C; `heap_5` rozszerza mechanikę `heap_4` na wiele regionów. Nie są to tematy tej lekcji.

1 Most: od K&R 5.4 do heap_4

K&R 5.4 — co mamy	Ograniczenie	Co dodaje heap_4
allocbuf[64], allocp wynik to surowy adres	istnieje tylko wolny ogon areny później nie znamy rozmiaru ob- szaru	arena podzielona na opisane bloki nagłówki BlockLink_t przed payloadem
przydział przesuwaj kursor	nie wykorzystamy dziury	lista wolnych bloków i first-fit
reset całej areny	brak zwolnienia jednego ob- szaru	vPortFree() wskazanego bloku
brak relacji między dziurami	sąsiednie dziury są rozdzielone	porządek adresowy i coalescing
sukces albo NULL	brak miary fragmentacji	free, largest, block count, minimum-ever

Nagłówek realnego bloku

```
typedef struct A_BLOCK_LINK {  
    struct A_BLOCK_LINK *pxNextFreeBlock;  
    size_t xBlockSize;    /* MSB oznacza blok przydzielony */  
} BlockLink_t;  
  
[ BlockLink_t | payload zwracany przez pvPortMalloc() ]  
^ metadata      ^ adres widziany przez aplikację
```

Granica modelu: Task 1–2 używają uproszczonego nagłówka {next,size}. Bit allocated i pełne zabezpieczenia oglądamy dopiero w upstream heap_4.c.

2 Demo 1 — pierwszy pasujący blok i split (5 minut)

Na RP2350/RV32 nagłówek modelu ma 8 B. Dla żądania 13 B:

$$\text{wanted} = \text{align8}(8 + 13) = 24 \text{ B}$$

```
free list przed: [ block 16 B ] -> [ block 256 B ] -> END  
request po nagłówku i align8: 24 B
```

Predykcja — wpisz przed odsłonięciem:

- Pierwszy pasujący blok ma _____ B.
- Blok przydzielony ma _____ B, a remainder _____ B.
- Równość zachowania areny: _____ + _____ = 256 B.

Sprawdzenie po predykcji: 16 B nie mieści żądania, więc first-fit wybiera 256 B; split daje 24 B + 232 B. Realny heap_4 dzieli blok tylko wtedy, gdy reszta jest większa od minimalnego rozmiaru bloku.

Nie uruchamiamy osobnego debugowania Task 1. Model służy do policzenia geometrii; centralny RUN jest w Task 3.

3 Demo 2 — lista adresowa i coalescing (7 minut)

[A:48 FREE][B:48 USED][C:48 FREE]

```
pamiec:  A ----- B ----- C
free list: A -----> C -> END
```

Predykcja: po `vPortFree(B)` narysuj listę i uzupełnij:

Stan	liczba wolnych bloków	suma wolna	największy blok
przed zwolnieniem B	_____	_____	_____
po zwolnieniu B	_____	_____	_____

Odsłoń po zapisaniu predykcji

1. Wstaw B między A i C według adresu.
2. `end(B) == address(C)`: połącz B z C.
3. `end(A) == address(BC)`: połącz A z BC.

```
przed: A:48 -> C:48 -> END    blocks=2 total=96 largest=48
po:    A:144 -> END           blocks=1 total=144 largest=144
```

Porządek po adresie sprawia, że fizycznych sąsiadów można sprawdzić podczas jednego wstawienia. To jest mechanizm funkcji `prvInsertBlockIntoFreeList()` z `heap_4.c`.

4 Task 3 — jedyny centralny RUN (11 minut)

Obraz i sesja muszą być przygotowane przed lekcją. Nie kompilujemy ani nie flashujemy w czasie tych 30 minut.

```
# wykonane przed lekcja:
stemctl debug rp2350 freertos heap4 3 --device /dev/bus/usb/BBB/DDD
```

```
RESET/INIT
-> ALLOC A(24) -> ALLOC B(40) -> ALLOC C(16)
-> FREE A -> FREE C
-> heap4_fragmented_checkpoint      # STOP 1
-> FREE B -> OOM request
-> task03_debug_checkpoint          # STOP 2
```

- STOP 1:** zapisz liczbę bloków, sumę i largest. Przed F5 przewidź, co zmieni `FREE B`.
- STOP 2:** sprawdź odzyskanie areny, minimum-ever, OOM i hook.

5 Dowód na Pico 2 W / RP2350

Ważne: heap_4 jest plikiem wybranym i linkowanym przez projekt. Nie jest sprzętowym heapem ani peryferium RP2350. Pico 2 W dostarcza realną pamięć SRAM, w której obserwujemy ten sam algorytm.

Minimalny widok GDB

```
p g_fragmented_stats
p g_final_stats
p xFreeBytesRemaining
p xMinimumEverFreeBytesRemaining
p xStart
p pxEnd
x/96bx ucHeap
```

Tabela obserwacji

Wielkość	Odczyt	Warunek
initial_free	_____	punkt odniesienia
after_allocations	_____	mniejsze niż initial
STOP 1: free blocks	_____	równe 2
STOP 1: total / largest	_____	largest mniejsze niż total
STOP 2: final free	_____	równe initial
STOP 2: free blocks	_____	równe 1
minimum-ever	_____	nie rośnie po free
OOM / hook / asserts /	_____	1 / 1 / 0 / 1
pass		

Wyjście — trzy minuty

1. Dlaczego jeden allocp nie wystarcza do zwolnienia obszaru spod offsetu 0 przy zachowaniu obszaru spod offsetu 5?

2. Jak vPortFree() znajduje rozmiar payloadu, skoro dostaje tylko jego adres?

3. Dlaczego 96 wolnych bajtów w dwóch blokach nie pozwala przydzielić jednego bloku 80 B?

Zaliczenie

- ☐ wymieniam: header, free list, coalescing;
- ☐ przewiduję 2 bloki -> 1 blok;
- ☐ odróżniam payload, metadata allocator'a i stos sp;
- ☐ odróżniam current free, largest free i minimum-ever;
- ☐ Task 3 kończy się aligned=1 oom=1 hooks=1 asserts=0 pass=1.

Poza lekcją: pełne czytanie upstream heap_4.c, osobne uruchomienie Task 1–2 na AMD64/Hazard3, samodzielny deploy RP2350 oraz porównanie z heap_1, heap_2, heap_3 i heap_5.