QXD0145 - Sistemas de Tempo-Real

Gerenciamento de Memória *heap* II

André Ribeiro Braga

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

21/03/2023

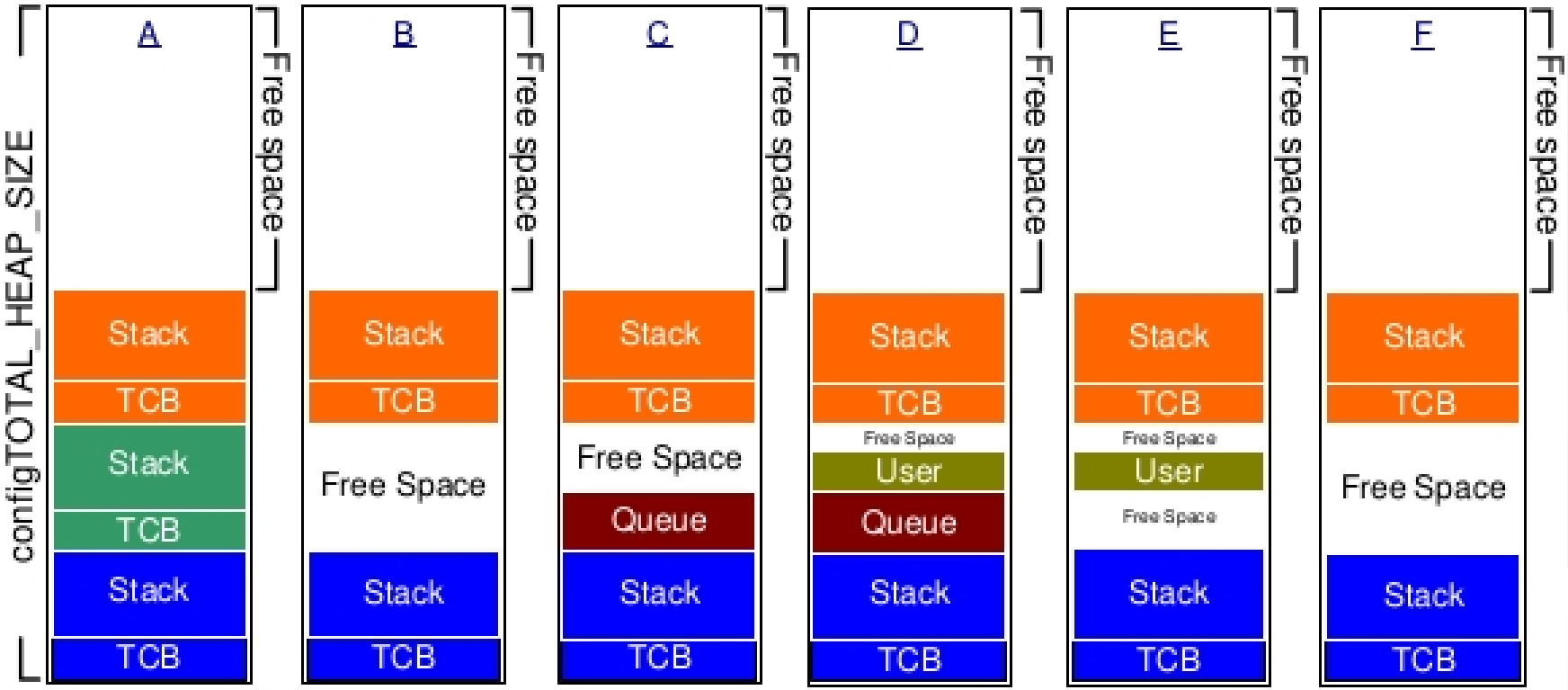
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_4

- Subdivide o arranjo estático em blocos menores
- Combina blocos livres adjacentes
  - Minimiza fragmentação
- Algoritmo *first fit*
  - Primeiro bloco grande o suficiente
- Exemplo:
  - Existem três blocos livres com tamanhos 5, 200 e 100
  - *pvPortMalloc()* requisita 20 bytes de RAM
  - Bloco de tamanho 200 é dividido (20 + 180)
- Não determinístico (mais rápido que implementação padrão)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_4



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_4

Endereço de início do arranjo

- Por padrão, é declarado no fonte *heap_4.c*
  - Endereço decidido pelo *linker*
- Pode ser necessário posicionar o arranjo em lugar específico
- *configAPPLICATION_ALLOCATED_HEAP* ⇒ aplicação declara
- Declaração na aplicação ⇒ Pode definir endereço
- Sintaxe para posicionar arranjo depende do compilador
- Exemplos:
  - GCC: `uint8_t ucHeap[ configTOTAL_HEAP_SIZE ] __attribute__ ( ( section( ".my_heap" ) ) );`
  - IAR: `uint8_t ucHeap[ configTOTAL_HEAP_SIZE ] @ 0x20000000;`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_5

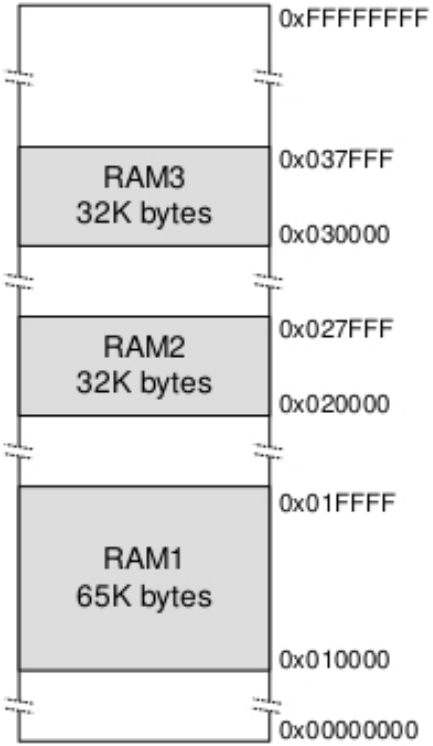
- Mesmo algoritmo de alocação do *heap_4* (*first fit*)
- Não limitado a um único arranjo estático
- Pode alocar múltiplos espaços de memória separados
- Útil para sistemas com múltiplas memórias não contínuas
- Único esquema de alocação que requer inicialização explícita
  - `void vPortDefineHeapRegions( const HeapRegion_t * const pxHeapRegions );`

```
typedef struct HeapRegion
{
    /* The start address of a block of memory that will be part of the heap.*/
    uint8_t *pucStartAddress;

    /* The size of the block of memory in bytes. */
    size_t xSizeInBytes;

} HeapRegion_t;
```

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_5

Exemplo

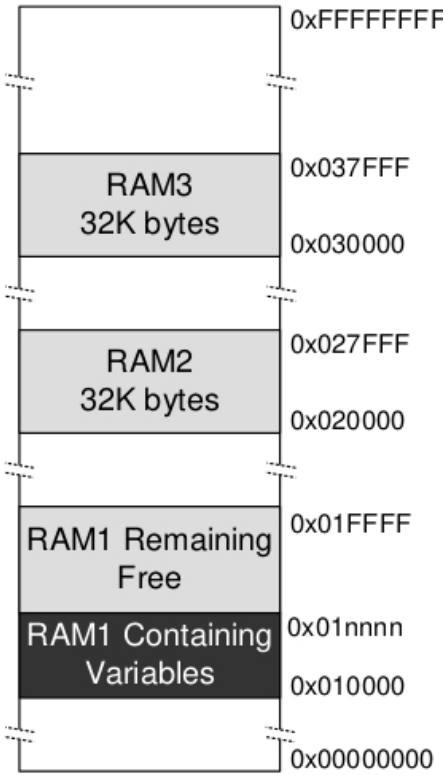



```
/* Define the start address and size of the three RAM regions. */
#define RAM1_START_ADDRESS    ( ( uint8_t * ) 0x00010000 )
#define RAM1_SIZE             ( 65 * 1024 )

#define RAM2_START_ADDRESS    ( ( uint8_t * ) 0x00020000 )
#define RAM2_SIZE             ( 32 * 1024 )

#define RAM3_START_ADDRESS    ( ( uint8_t * ) 0x00030000 )
#define RAM3_SIZE             ( 32 * 1024 )

/* Create an array of HeapRegion_t definitions, with an index for each of the three
RAM regions, and terminating the array with a NULL address.  The HeapRegion_t
structures must appear in start address order, with the structure that contains the
lowest start address appearing first. */
const HeapRegion_t xHeapRegions[] =
{
    { RAM1_START_ADDRESS, RAM1_SIZE },
    { RAM2_START_ADDRESS, RAM2_SIZE },
    { RAM3_START_ADDRESS, RAM3_SIZE },
    { NULL,               0         }  /* Marks the end of the array. */
};

int main( void )
{
    /* Initialize heap_5. */
    vPortDefineHeapRegions( xHeapRegions );

    /* Add application code here. */
}
```

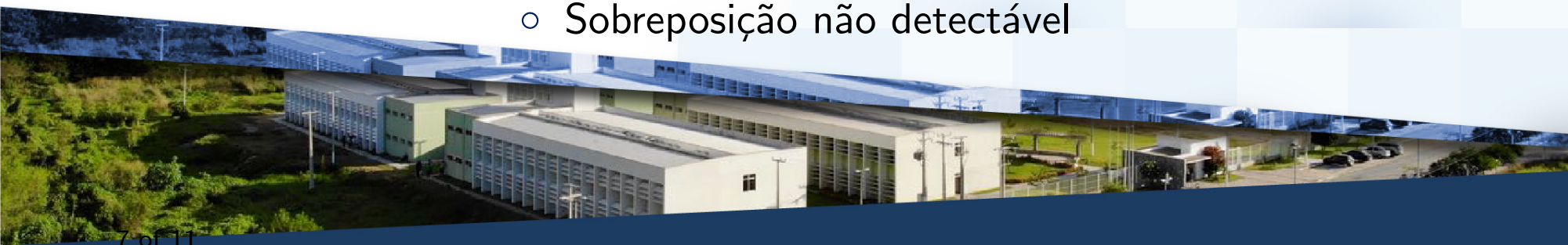
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_5

Exemplo



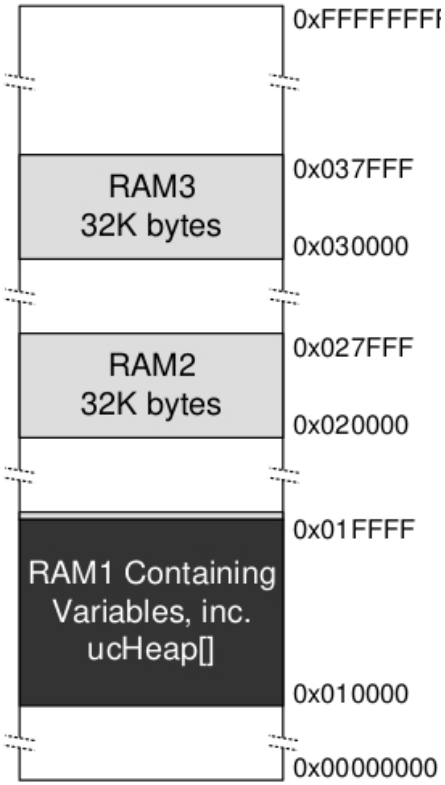
- Solução anterior não reserva espaço para as variáveis
- Durante o *build*, o *linker* aloca RAM
  - Arquivo de configuração do *linker*
- Figura assume essa configuração
  - *Linker* ignora RAM2/RAM3
  - Endereços acima de *0x0001nnnn* para o *heap*
  - Valor de *0x0001nnnn* depende das variáveis
  - Início do *heap* deve utilizar *0x0001nnnn*
- Não recomendável
  - *0x0001nnnn* difícil de determinar
  - Variável em futuras versões
  - Sobreposição não detectável



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_5

Exemplo




```
/* Define the start address and size of the two RAM regions not used by the
linker. */
#define RAM2_START_ADDRESS    ( ( uint8_t * ) 0x00020000 )
#define RAM2_SIZE             ( 32 * 1024 )

#define RAM3_START_ADDRESS    ( ( uint8_t * ) 0x00030000 )
#define RAM3_SIZE             ( 32 * 1024 )

/* Declare an array that will be part of the heap used by heap_5.  The array will be
placed in RAM1 by the linker. */
#define RAM1_HEAP_SIZE ( 30 * 1024 )
static uint8_t ucHeap[ RAM1_HEAP_SIZE ];

/* Create an array of HeapRegion_t definitions.  Whereas in Listing 6 the first entry
described all of RAM1, so heap_5 will have used all of RAM1, this time the first
entry only describes the ucHeap array, so heap_5 will only use the part of RAM1 that
contains the ucHeap array.  The HeapRegion_t structures must still appear in start
address order, with the structure that contains the lowest start address appearing
first. */
const HeapRegion_t xHeapRegions[] =
{
    { ucHeap,             RAM1_HEAP_SIZE },
    { RAM2_START_ADDRESS, RAM2_SIZE },
    { RAM3_START_ADDRESS, RAM3_SIZE },
    { NULL,               0         }  /* Marks the end of the array. */
};
```

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Funções Relacionadas do API

xPortGetFreeHeapSize()

- Retorna o número de *bytes* livres no instante da chamada
- Pode ser utilizado para otimizar o tamanho do *heap*
- Exemplo:
  - Retorna valor 2000 após todos os objetos criados
  - O valor de *configTOTAL_HEAP_SIZE* pode ser reduzido de 2000

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Funções Relacionadas do API

xPortGetMinimumEverFreeHeapSize()

- Retorna o valor mínimo de *bytes* livres durante execução
- Indica quão perto a aplicação chegou de esgotar o *heap*
- Apenas disponível para as opções *heap_4* e *heap_5*

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Funções Relacionadas do API

Malloc Failed Hook

- *pvPortMalloc* retorna *NULL* caso não haja sucesso
- Quando criando um objeto do *kernel*, ele não será criado
- Caso ocorra falha, uma função *hook* pode ser chamada
- *configUSE_MALLOC_FAILED_HOOK* deve ser configurado
  - Implementar *vApplicationMallocFailedHook*