QXD0145 - Sistemas de Tempo-Real

Gerenciamento de Memória *heap* I

André Ribeiro Braga

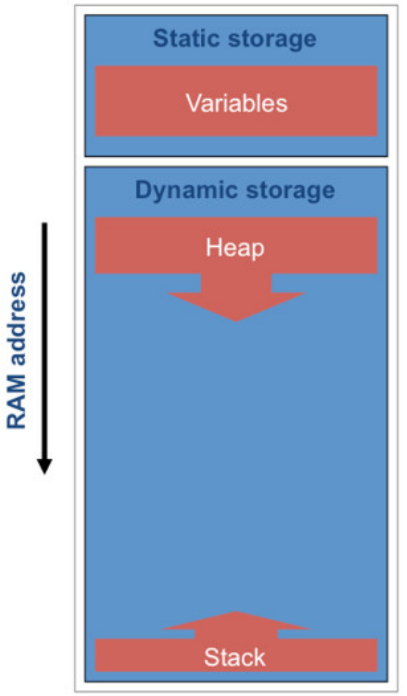
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

20/03/2023

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Alocação Dinâmica de Memória

- *Kernel* do SO estruturado em objetos
- Alocação em tempo de execução
  - Reduz esforço de projeto e planejamento
  - Simplifica a API
  - Minimiza o *footprint* de RAM
- Conceito de programação (C/C++)



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Alocação Dinâmica de Memória

Esquemas padrão nem sempre adequados

- Nem sempre disponíveis
- Implementação pode ser grande
- Raramente são *thread-safe*
- Não são determinísticos
- Podem sofrer de fragmentação
- Podem complicar a configuração do *linker*
- Possível fonte de erros de difícil depuração

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Alocação Dinâmica no FreeRTOS

Histórico

- Alocação com um *pool* de blocos (tamanhos diferentes)
- *Pools* pré-alocados em tempo de compilação
  - Retornado por funções de alocação
- Estratégia comum em sistemas de tempo-real
- Não utiliza a RAM de forma eficiente
  - Inviável para sistemas embarcados simples
- Agora faz parte da camada portável
  - Diferenças entre plataformas
  - Costumização por parte do desenvolvedor da aplicação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Alocação Dinâmica no FreeRTOS

Mecanismo Básico

- Quando o FreeRTOS requer RAM
  - *pvPortMalloc()* em vez de *malloc()*
- Quando o FreeRTOS libera RAM
  - *pvPortFree()* em vez de *free()*
- Funções públicas (chamadas do código da aplicação)
- Mesmos protótipos das funções padrão
- Existem 5 exemplos de alocação definidos
- Aplicações podem utilizar um deles ou implementar um específico
- Código no diretório *FreeRTOS/Source/portable/MemMang*

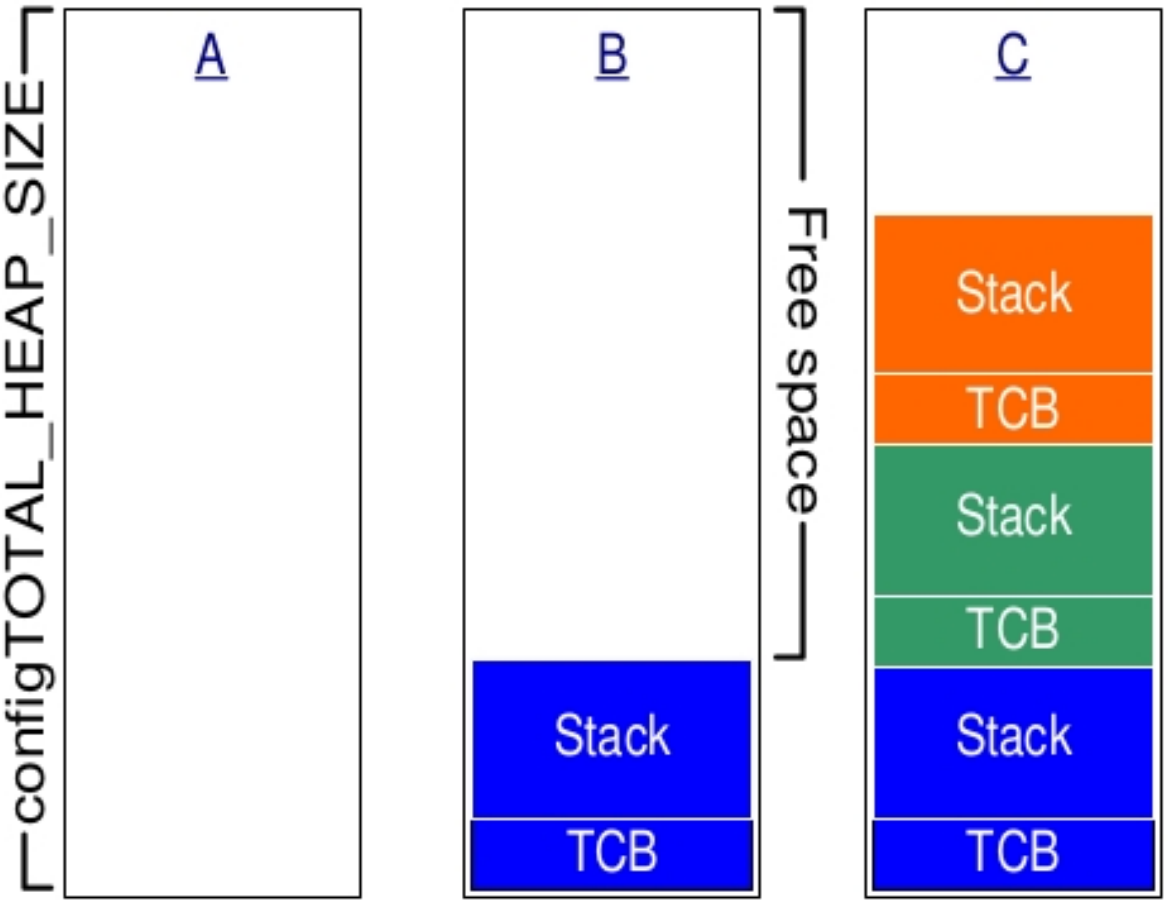
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_1

- Comum para pequenos sistemas embarcados
- Criação de objetos antes do início da aplicação
- Versão simples do *pvPortMalloc()*
- Não implementa o *pvPortFree()*
  - Aplicação não libera memória
- Alocação determinística e não fragmenta memória
- Sistemas em que alocação dinâmica é proibitiva
- Tamanho do *heap* definido por *configTOTAL_HEAP_SIZE*

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_1



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_2

- Mantido no *kernel* por retrocompatibilidade
- Uso não recomendado para novos projetos (usar *heap_4*)
- Também subdivide um arranjo com tamanho *configTOTAL_HEAP_SIZE*
- Permite que memória seja liberada
- Algoritmo *best fit*
  - Bloco com tamanho mais próximo da requisição

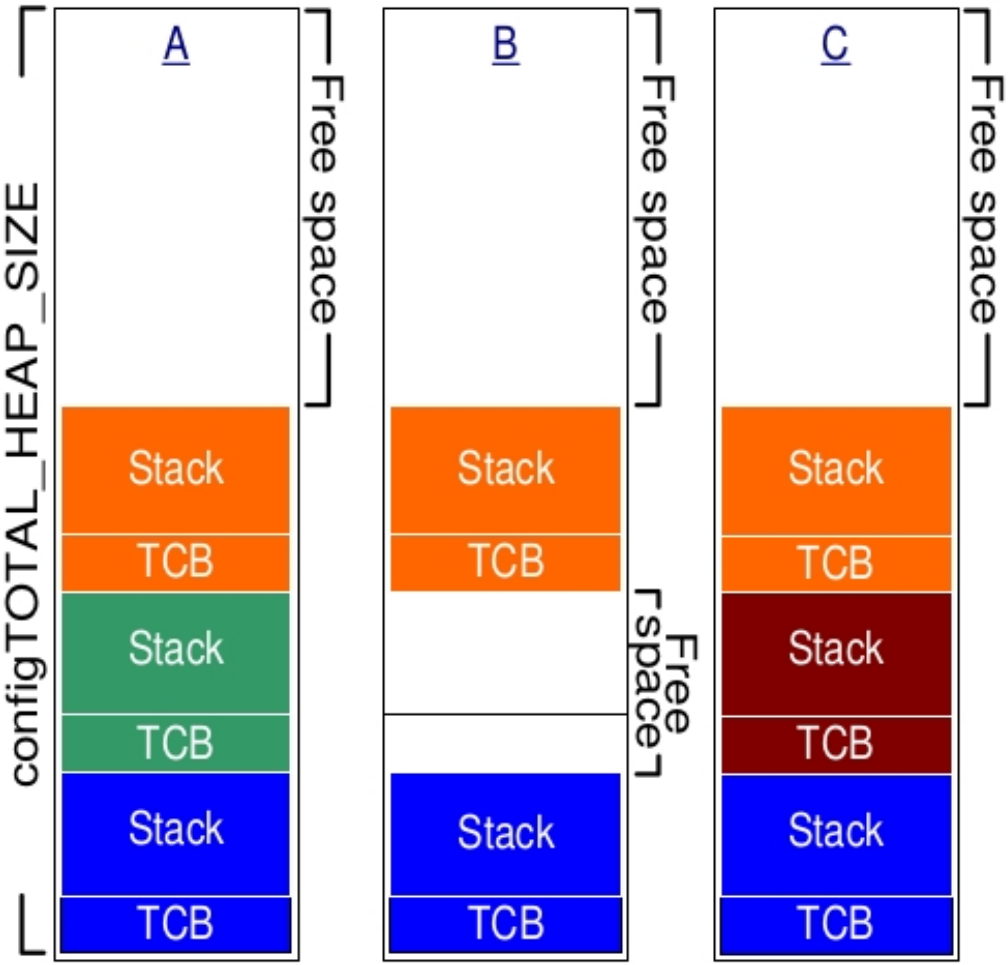
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_2

- Exemplo:
  - O *heap* contém 3 blocos livres com tamanhos 5, 25 e 100 bytes
  - Chamada do *pvPortMalloc* requer 20 bytes
  - Bloco de 25 bytes dividido em dois (20 + 5)
- Não combina blocos livres adjacentes
- Mais sucetível à fragmentação
- Não problemático se blocos alocados/liberados têm mesmo tamanho
- Não determinístico mas mais rápido que implementações padrão

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_2



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Heap_3

- Utiliza as chamadas padrão: *malloc()* e *free()*
- Tamanho do *heap* definido pelo *linker*
- *configTOTAL_HEAP_SIZE* não tem qualquer efeito
- *Thread-safe* através da suspensão do escalonador