IM420 Sistemas Embarcados de Tempo Real

Notas de Aula – Semana 11

Prof. Denis Loubach
dloubach@fem.unicamp.br

Programa de Pós-Graduação em Eng. Mecânica / Área de Mecatrônica
Faculdade de Engenharia Mecânica - FEM
Universidade Estadual de Campinas - UNICAMP

1^o Semestre de 2018

Tópicos

Taking the Internet to the Next Physical Level

The Internet of Things (IoT) is getting ready to explode. Researchers predict that the number of devices connected to the Internet will increase 10- to 100-fold in the next ten years. These devices run the gamut from personal wearables to global environmental sensor networks. They include connected security cameras, thermostats, and smoke alarms in smart homes, and ubiquitous sensors in smart cities that compile and respond to data about air pollution, waste management, crime prevention, and traffic congestion. In spite of the promise such technology holds, innovators are grappling with challenges including how complicated, insecure, and gimmicky IoT devices often seem to the general public

Fonte: **IEEE Computer, February 2016**
`https://www.computer.org/csdl/mags/co/2016/02/mco2016020080.html`

Visão Geral

Geralmente, implementa-se um gerenciamento de memória customizado no topo das facilidades que o RTOS provê

O entendimento do gerenciamento de memória em sistemas embarcados é considerado fundamental para o seu desenvolvimento

Por exemplo, em vários sistemas embarcados existe a chamada de uma rotina para alocação dinâmica de memória (`malloc`)

Entretanto, a utilização desta rotina pode causar um efeito colateral não desejado denominado **fragmentação de memória**

Visão Geral (cont...)

Essa rotina genérica de alocação dinâmica de memória, dependendo de sua implementação, pode causar impacto (negativo) na performance da aplicação

Além disso, pode ainda não suportar a política de alocação de memória requerida pela aplicação

Independente do tipo de sistema embarcado, alguns requisitos são comuns para a questão de gerenciamento de memória:

- Fragmentação mínima
- *Overhead* de gerenciamento mínimo
- Tempo de alocação determinístico

Visão Geral (cont...)

O foco desta aula concentra-se nos seguintes tópicos:

- Fragmentação de memória
- Compactação de memória
- Alocação dinâmica
- Gerenciamento de memória de tamanho fixo
- Unidade de hardware de gerenciamento de memória (*memory management unit - MMU*)

Visão Geral (cont...)

Para os aspectos de gerenciamento de memória examinados no escopo desta aula, utiliza-se exemplos de implementações para as funções `malloc` e `free` para sistemas embarcados

Alocação dinâmica de memória

Conforme visto em aulas anteriores, sabemos quais as regiões de memória física são ocupadas após a inicialização do sistema:

- Código do programa (`.text` executado da ROM ou Flash)
- Dados do programa (`.data` copiado para RAM e `.bss` reservado na RAM)
- Pilha do sistema (`stack` reservado na RAM)

Neste ponto, tanto um RTOS quanto um *kernel* utilizam o espaço de memória física restante para **alocação dinâmica de memória**

Esta área de memória física é denominada ***heap***

O gerenciamento de memória no contexto abordado em [1] refere-se ao gerenciamento de blocos contíguos

Alocação dinâmica de memória (cont...)

Em geral, uma *memory management facility* mantém informações internas para o *heap* numa área de memória reservada chamada *control block*

Informações típicas para este *control block* incluem:

- Endereço de início do bloco de memória física utilizado para alocação dinâmica de memória
- Tamanho do bloco de memória física
- Tabela de alocação que indica quais áreas de memória encontram-se em uso, quais estão livres, e qual o tamanho das regiões livres

Fragmentação de memória

Neste exemplo de fragmentação, o *heap* é quebrado em blocos pequenos de tamanho fixo

Cada bloco possui uma unidade de tamanho que é uma potência 2

Essa quebra visa facilitar a tradução de um tamanho requisitado num número de unidades correspondente ao tamanho requisitado

Neste exemplo, utiliza-se 32 bytes como tamanho de uma unidade

Fragmentação de memória (cont...)

A função da alocação dinâmica de memória `malloc` possui um parâmetro de entrada que especifica o tamanho da alocação requerida em bytes

Esta função aloca um bloco maior que é constituído de um ou mais dos blocos menores e de tamanho fixo

O tamanho deste bloco de memória maior é no mínimo tão grande quanto o tamanho requisitado

É ainda o múltiplo mais próximo do tamanho de uma unidade

Fragmentação de memória (cont...)

Por exemplo: se uma alocação requisita 100 bytes, o bloco retornado é de 128 bytes. Ou seja, 4 unidades $\times$ 32 bytes/unidade

Como resultado, quem requisitou a alocação não utilizará 28 bytes da memória alocada

Este fenômeno é conhecido como **fragmentação de memória**

Este exemplo específico é conhecido como **fragmentação interna** uma vez que a fragmentação é interna ao bloco alocado

Fragmentação de memória (cont...)

A tabela de alocação pode ser representada como um *bitmap* em que cada bit representa uma unidade de 32-byte

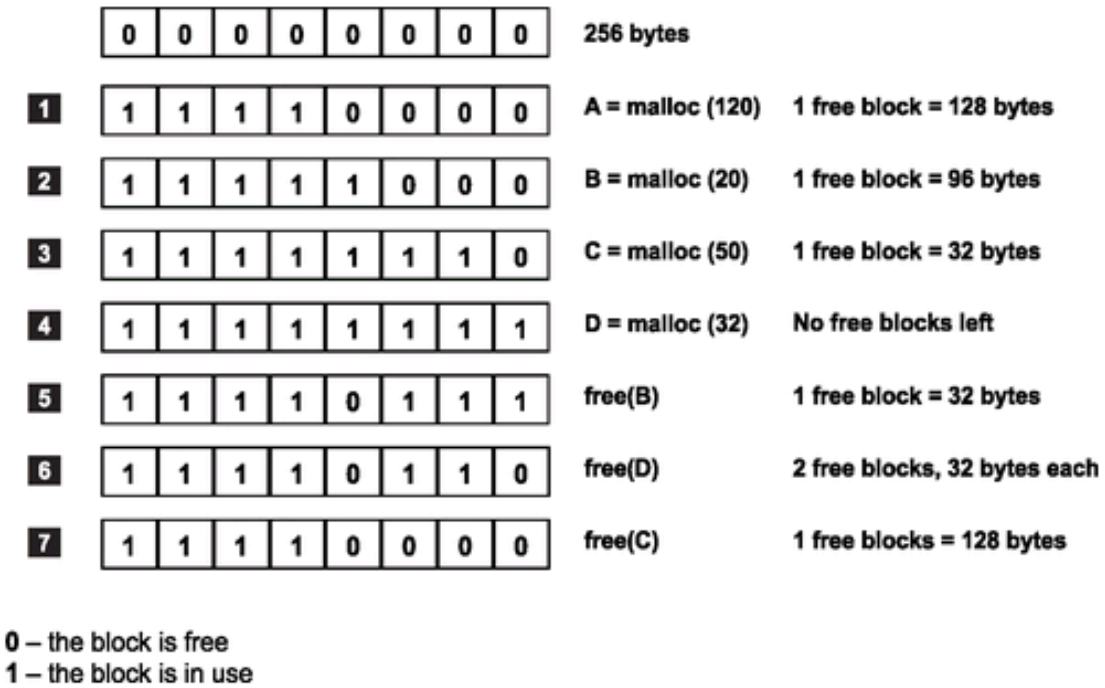

Figura: Estados de um mapa de alocação de memória, *heap* de 256 bytes

Fragmentação de memória (cont...)

O passo 6 mostra 2 blocos livres de 32 bytes cada

O passo 7, entretanto, mostra que os blocos livres (32-byte) foram combinados para formar um bloco livre de 128-byte

Dado que esses blocos foram combinados, uma próxima alocação de 96 bytes deve ser bem sucedida

Fragmentação de memória (cont...)

A figura a seguir mostra outro exemplo de estados de uma tabela de alocação

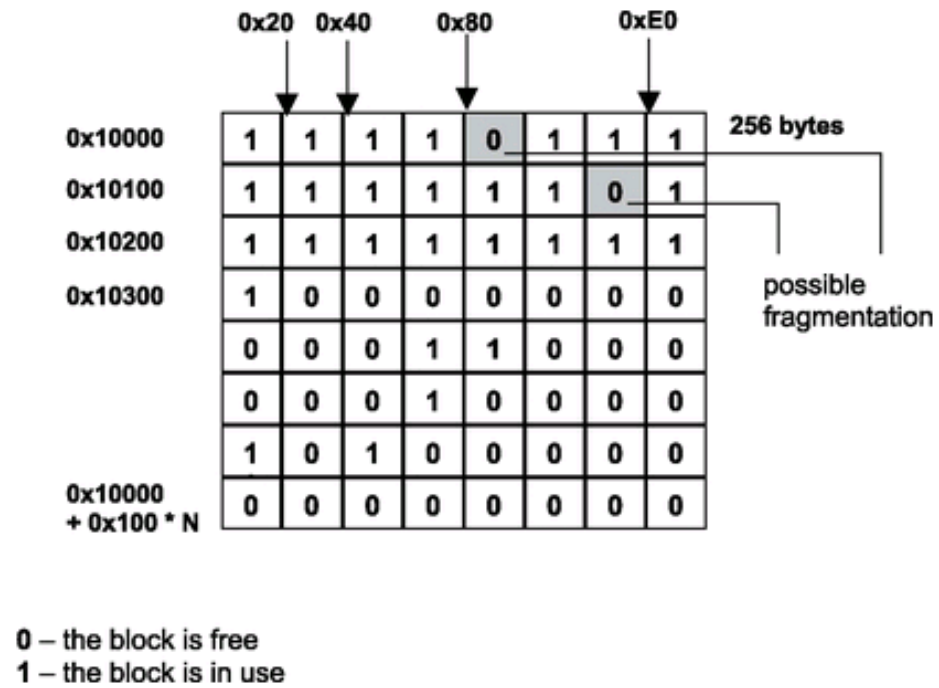

Figura: Mapa de alocação de memória com possível fragmentação

Fragmentação de memória (cont...)

Note que na figura anterior existem dois blocos livres de 32-byte cada

Um bloco no endereço `0x10080` e outro bloco no endereço `0x101C0`

Estes blocos não podem ser combinados pois não estão contíguos

Portanto, nenhuma alocação de mais que 32 bytes será bem sucedida, mesmo o sistema apresentando 64 bytes livres

A existência destes dois blocos isolados é considerada **fragmentação externa**, porque a fragmentação existe na tabela em si, e não nos blocos

Compactação de memória

Uma possibilidade de se eliminar o problema da fragmentação é através da **compactação de memória**

No exemplo anterior, isso seria realizada através da compactação da área adjacente dos dois blocos isolados

O conteúdo do intervalo de memória a partir do endereço `0x100A0` (imediatamente após o primeiro bloco livre) até o endereço `0x101BF` (imediatamente antes do segundo bloco livre) é deslocado 32 bytes a esquerda na memória

Isto resultaria no novo intervalo de `0x10080` até `0x1019F`

O que efetivamente combina dois blocos livres de memória formando um bloco de 64-byte

Compactação de memória (cont...)

Entretanto, a compactação de memória continua até que todos os blocos livres sejam combinados em um grande bloco livre

Neste caso, vários problemas podem ocorrer com a compactação de memória:

- A transferência de conteúdo de uma localização para outra é considerada *time-consuming*
- O custo da operação de cópia depende do tamanho dos blocos contíguos em uso
- As tarefas que detêm a propriedade das áreas de memória que serão manipuladas não poderão acessar o conteúdo dessa memória até que a operação de transferência se complete

Compactação de memória (cont...)

Na prática, compactação de memória é quase nunca realizada em sistemas embarcados

Os blocos livres de memória somente são combinados se eles forem vizinhos imediatos (como visto no primeiro exemplo)

Compactação de memória (cont...)

Outras restrições para utilização de compactação de memória em sistemas embarcados:

- Compactação de memória é permitida se as tarefas que possuem blocos de memória afetados fizerem referência aos tais blocos através de endereçamento virtual. Se as tarefas utilizarem o endereço físico diretamente esta operação não é permitida
- Em muitos casos, o sistema de gerenciamento de memória também deve considerar os requisitos de alinhamento da memória que são específicos de arquitetura
- Alinhamento de memória refere-se as restrições específicas de arquitetura impostas ao endereço de um dado na memória
- Por exemplo, algumas arquiteturas requerem que dados multi-byte (*integers* e *long integers*) sejam alocados em endereços que são potência de 2. Endereços de memória desalinhados podem resultar em exceções como *bus error*

Observações quanto ao gerenciamento eficiente de memória

- Determinar se um bloco livre e suficientemente grande existe para satisfazer uma requisição de alocação (parte do trabalho do `malloc`)
- Atualizar a informação de gerenciamento interno (parte do trabalho do `malloc` e `free`)
- Determinar se os blocos recém liberados podem ser combinados com seus vizinhos para formar um bloco maior (parte do trabalho do `free`)

Sobre a tabela de alocação

A estrutura da tabela de alocação é a chave para um gerenciamento de memória eficiente

Isso porque a sua estrutura determina como as operações listadas anteriormente devem ser implementadas

A tabela de alocação é parte do *overhead* uma vez que ela ocupa um espaço de memória que é excluído do uso do software aplicativo

Consequentemente, outro requisito é reduzir o *overhead* de gerenciamento

Gerenciamento de memória de tamanho fixo

Uma outra abordagem para o gerenciamento de memória é a utilização do método de *pools* de memória com blocos de tamanho fixo

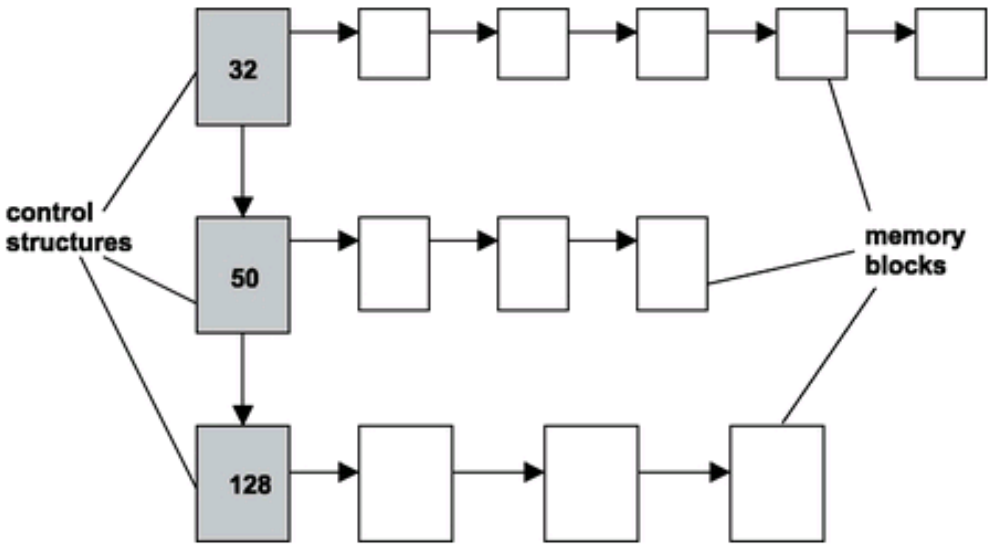

Figura: Gerenciamento de memória baseado em *pools* de memória

Gerenciamento de memória de tamanho fixo (cont...)

Como mostrado na figura anterior, o espaço de memória disponível é dividido em vários *pools* de memória de tamanho fixo diferentes (três *pools* com tamanho de blocos de 32, 50, e 128)

Todos os blocos num mesmo *pool* de memória possuem o mesmo tamanho

Cada estrutura de controle de *pool* de memória mantém informações como:

- Tamanho do bloco
- Número total de blocos
- Número de blocos livres

Neste exemplo anterior, os *pools* de memória são ligados juntos e ordernados por tamanho

Gerenciamento de memória de tamanho fixo (cont...)

Encontrar o menor tamanho adequado para uma alocação requer procurar nesta **lista-ligada** e examinar cada estrutura de contorle para o primeiro tamanho de bloco adequado

Uma alocação com sucesso resulta numa entrada sendo **removida** do *pool* de memória

Uma desalocação com sucesso resulta numa entrada sendo **inserida** de volta no *pool* de memória

Considerações sobre gerenciamento de memória de tamanho fixo

Em vários casos, os *pools* de memória são construídos baseados em suposições

Isto pode resultar em sub-uso ou mesmo não uso dos *pools* de memória

Em outros casos, também pode acontecer o sobre-uso

Por outro lado, este método pode contribuir para redução da fragmentação interna e prover uma alta utilização para sistemas embarcados estáticos

Sistemas embarcados estáticos são aqueles que possuem ambiente predizível (número conhecido de tarefas em execução na inicialização e tamanho de bloco de memória conhecidos inicialmente)

Considerações sobre gerenciamento de memória de tamanho fixo (cont...)

Este método é considerado mais determinístico em relação ao método do algoritmo de *heap*

No método de *heap*, cada operação de `malloc` ou `free` pode potencialmente disparar um re-arranjo no *heap*

No método de *pool* de memória, os blocos de memória são tomados e retornados no início da lista-ligada, então as operações gastam um tempo constante

Além disso, *pool* de memória não requer re-estruturação

Unidade de gerenciamento de memória (hardware)

Até este ponto, discutiu-se a questão de gerenciamento de memória física

Um outro tópico, é o gerenciamento de **memória virtual**

Memória virtual é a técnica na qual uma memória de armazenamento de massa (*e.g., hard disk*) aparece para a aplicação como se fosse RAM

O espaço de endereçamento virtual de memória (também chamado de espaço de endereçamento lógico) é maior que o espaço de memória física real

Neste caso, a unidade de **gerenciamento de memória (hardware)** (ou *memory management unit - MMU*) figura para prover várias funções

MMU (cont...)

Primeiro, a MMU traduz o endereço virtual para um endereço físico a cada acesso à memória

Segundo, a MMU provê proteção à memória

A função de tradução de endereços difere de um projeto de MMU para outro

Além disso, muitos RTOS não suportam a implementação de endereçamento virtual

MMU (cont...)

Se a MMU estiver habilitada num sistema embarcado, a memória física é geralmente dividia em **páginas**

Um conjunto de atributos é associado a cada página de memória:

- se a página contém código (*e.g.,* instruções executáveis) ou dado
- se a página pode ser escrita, lida ou executada (ou a combinação destes atributos)
- se a página pode ser acessada quando a CPU não estiver no modo de execução privilegiado

MMU (cont...)

Quando a MMU encontra-se habilitada, todo acesso à memória é realizado pela MMU

Neste caso, o hardware força o acesso à memória de acordo com os atributos das páginas

Por exemplo, se uma tarefa tenta escrever numa região de memória que permite apenas leitura, a operação é considerada ilegal, e portanto a MMU não a permite

O resultado disso é que esta operação dispara uma exceção de acesso à memória

Informação ao leitor

Notas de aula baseadas nos seguintes textos:

Q. Li and C. Yao, *Real-Time Concepts for Embedded Systems*.
CMP books, 2003.