Estruturas de Dados I - Lista Encadeada
1ª Exercício Computacional
Sistemas de Informação/CPCX - UFMS
30/05/2019

Simulador de Alocação de Memória

1. Introdução

Você já se perguntou como é gerenciado o processo de alocação de memória em tempo de execução? Ou até mesmo, quem é o responsável por esse gerenciamento? Provavelmente que sim.

Quando o seu programa utiliza o comando **new** (ou equivalente), solicitando uma quantidade n de células de memória, o sistema operacional ativa um processo a fim de encontrar um bloco (células consecutivas) disponível de tamanho n, marcar as células deste bloco como ocupadas e subtrair n do número total de células disponíveis. De modo análogo, quando seu programa utiliza o comando **delete** (ou equivalente), devolvendo uma quantidade m de células de memória, o sistema operacional ativa um processo que marca estas células como disponíveis e adiciona m ao total de células disponíveis.

Para entender como é realizado o gerenciamento do processo de **solicitação** e **devolução** de células de memória, duas questões fundamentais devem ser respondidas:

1. Como o espaço de memória disponível é representado pelo sistema operacional?

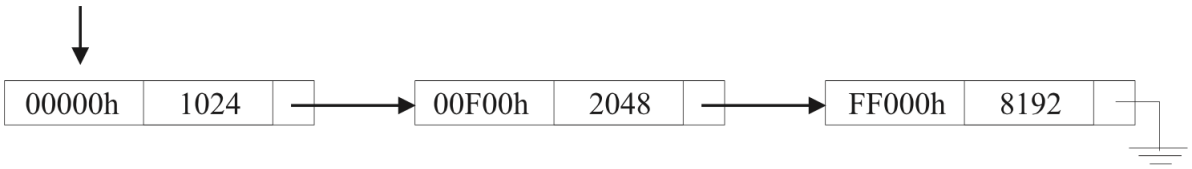
2. Dada uma representação para o espaço de memória disponível, durante o processo de alocação de memória, qual é o algoritmo que encontra um bloco de tamanho n?

Uma possível resposta para a questão (1) é uma lista linear ordenada simplesmente encadeada, onde cada nó da lista corresponde a um bloco de memória disponível e contém as seguintes informações: o endereço da primeira célula do bloco, a quantidade de células do bloco e um apontador para o próximo da lista. Os nós são dispostos, na lista, na ordem crescente do valor do campo endereço da primeira célula do bloco. Como exemplo, considere uma memória de 1 megabyte com células de 1 byte, contendo os seguintes blocos disponíveis:

Endereço da primeira célula	Número de células	Endereço do próximo bloco
00000h	1024	00F00h
00F00h	2048	01C00h
...	...	...
FF000h	8192	NULL

A lista linear contendo os blocos disponíveis acima é apresentada na figura abaixo. Observe que o primeiro bloco disponível é constituído pelas células compreendidas entre os endereços

`00000h` e `00400h`. As células compreendidas entre os endereços `00401h` e `00EFFh` estão ocupadas. O último bloco disponível se inicia no endereço `FF000h`.



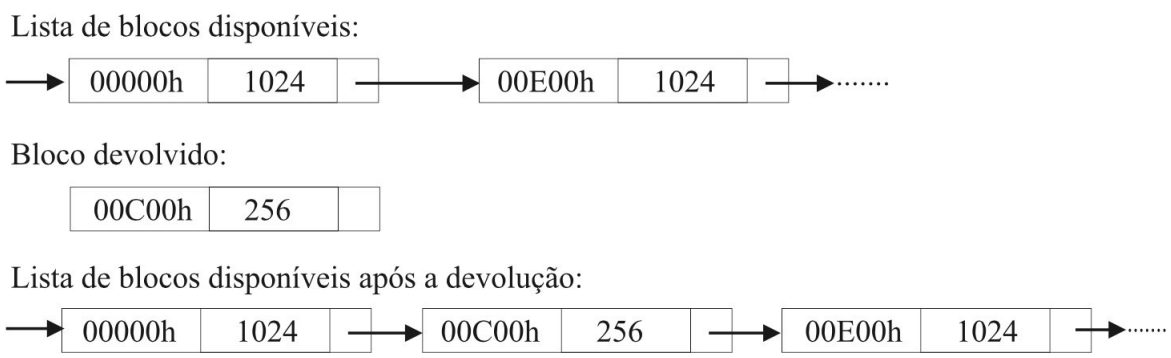
Agora é perfeitamente possível responder a questão (2). Se o seu programa solicita um bloco de memória de tamanho n, o sistema operacional procura na lista de blocos disponíveis um bloco de tamanhom, $m \geq n$, e reduz o tamanho do bloco para $m - n$. No caso $m = n$, o nó correspondente ao bloco é retirado da lista. Como pode haver na lista muitos blocos disponíveis com tamanho $m \geq n$, qual deles escolher? A maioria dos sistemas operacionais modernos que utilizam lista lineares para representarem os blocos disponíveis optam pelo método conhecido como *first fit*, isto é, escolher o primeiro bloco da lista com tamanho $m \geq n$. Um outro método conhecido como *best fit* foi muito utilizado pelos antigos sistemas operacionais. Este método consiste em escolher o bloco disponível cujo tamanho m é o menor dentre os demais blocos disponíveis com tamanho $m \geq n$.

2. O Simulador

Considere um computador com memória principal de 4 Kbytes cujas células possuem 1 byte cada. Então, construa um simulador para gerenciar o processo de solicitação e devolução de blocos de memória para este computador, sendo que o espaço de memória disponível deve ser representado por uma lista linear ordenada simplesmente encadeada contendo apenas as informações citadas na seção anterior. Os nós da lista devem estar ordenados pelo valor do campo endereço da primeira célula do bloco. O método para escolha de blocos disponíveis deve ser o *first fit*. O simulador deve aceitar como entrada solicitações ou devoluções de blocos de memória. Cada solicitação requer uma busca na lista de blocos disponíveis. Se n células são solicitadas e um bloco de $m > n$ células é encontrado, o campo quantidade de células do nó correspondente ao bloco encontrado é atualizado para $m - n$. Se um bloco de $m = n$ células é encontrado, o nó correspondente ao bloco encontrado deve ser retirado da lista. Se nenhum bloco disponível de tamanho $m \geq n$ é encontrado na lista, a mensagem "não há memória suficiente" deve ser apresentada no vídeo. A operação de devolução é um pouco mais complicada, pois uma das três situações podem ocorrer: um nó deve ser inserido na lista, um nó já existente deve ser atualizado ou dois nós já existentes são unidos para gerar um novo nó.

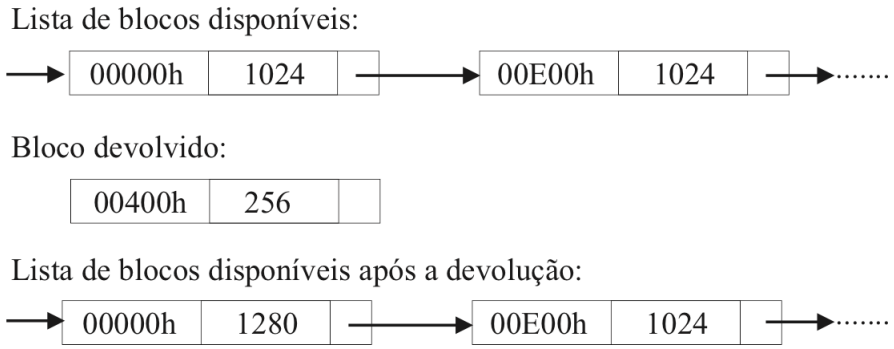
A primeira situação ocorre quando o bloco devolvido não faz fronteira com nenhum dos blocos representados na lista.

Bloco devolvido não faz fronteira com nenhum dos outros blocos da lista



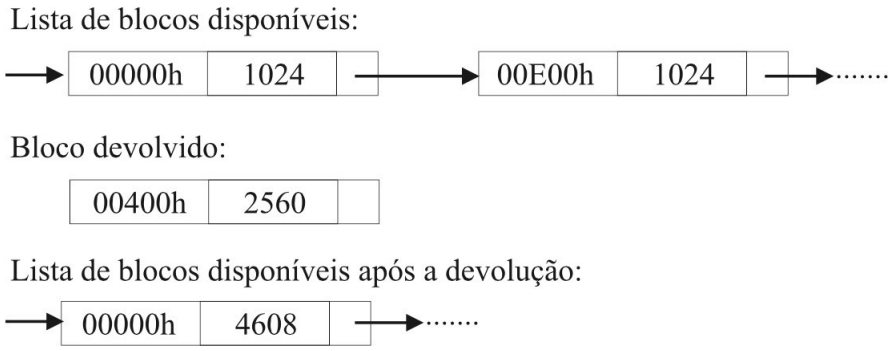
A segunda situação ocorre quando o bloco disponível faz fronteira com apenas um bloco representado na lista.

Bloco devolvido faz fronteira com um dos blocos da lista



A terceira situação ocorre quando o bloco devolvido faz fronteira com dois blocos representados na lista.

Bloco devolvido faz fronteira com dois blocos da lista



3. Entrada e Saída do programa:

Entrada:

Em cada linha seu programa deve ler um inteiro N que indica uma das 4 opções:

1. - Solicitação de memória
2. - Devolução de memória
3. - Impressão dos Blocos de Memória Livres
4. - Impressão dos Blocos de Memória Ocupados

Seu programa deve parar quando ler $N == 0$. Quando $N == 1$ ou $N == 2$, seu programa deve ler na mesma linha um inteiro T que indica o número de células a ser solicitado/devolvido. Para $N == 2$, seu programa deve ler na mesma linha um terceiro valor em hexadecimal E indicando o endereço de memória em que se inicia o bloco a ser devolvido.

Saída:

Seu programa deve imprimir a Lista de Blocos de Memória Livre e/ou a Lista de Blocos de Memória Ocupada quando for solicitado e deixar uma linha em branco após a impressão de cada lista.

Restriçoes:
$0 \leq N \leq 4$
$1 \leq T \leq 4096$
$0 \leq E \leq FFF$

Modelo de Entrada e Saída:

Entrada	**Saída**
1 4096 2 512 400h 3 2 2460 664h 1 2000 3 2 16 600h 4 0	Memória Livre: ↦ [400h , 512] ↦ Memória Livre: ↦ [400h , 512] ↦ [e34h , 460] ↦ Memória Ocupada: ↦ [0h , 1024] ↦ [610h , 2084] ↦

3. Prazo de Entrega

O prazo para submissão dos trabalhos, sem prejuízo na nota, encerra-se no dia 14/06/2019 às 23h59m. Os trabalhos entregues fora do prazo receberão nota zero e não serão corrigidos.

4. Normas para Submissão do Trabalho

1. A entrega deve ser feita, EXCLUSIVAMENTE, via AVA-UFMS.
2. Todos os arquivos utilizados no projeto devem ser entregues em ARQUIVO ÚNICO COMPACTADO com extensão .rar. Não se esqueçam de incluir o nome do autor do projeto no código fonte.
3. O arquivo compactado com os fontes deverá incluir também um arquivo chamado LEIA-ME, em formato TEXTO (txt), com as seguintes informações:
   (a) Nome completo do autor;
   (b) Arquivos fontes;
   (c) Dicas de utilização do programa (como interagir com o programa);
   (d) Breve descrição dos componentes do software (organização dos fontes, classes, métodos, módulos, etc);
   (e) Observações (destacar pontos positivos e justificar pontos negativos);
4. Não se esqueçam de verificar se o arquivo foi enviado corretamente. Os arquivos que não puderem ser abertos ou foram enviados de forma incompleta ou incorreta não serão corrigidos e receberão nota 0 (zero). A avaliação dos trabalhos será realizada em um ambiente Windows/NetBeans.

5. Critério para Avaliação do Trabalho

O critério de avaliação levará em conta dois fatores:

1. Entrega, organização do código, do material de apoio e documentação (3.0 pontos), incluindo os aspectos de:

   - Clareza, consistência, abrangência e organizacão do material entregue, incluindo o arquivo LEIA-ME, e da documentação dos arquivos que contém o código-fonte do projeto.
   - Portabilidade e facilidade de execução do programa.
   - Organização e estruturação das classes e/ou rotinas, utilização adequada de variáveis e estruturas de dados, separação em módulos consistentes com a especificação do problema, facilidade para reaproveitamento de código.

2. Corretude do programa (7.0 pontos). No caso da corretude do seu programa, a nota atribuída ao programa será proporcional ao número de casos de teste para os quais o seu programa forneceu a resposta correta. Há 07 (sete) casos de teste. A nota atribuída ao seu programa será proporcional a n pontos, onde n é o número de casos de teste para os quais o programa forneceu a saída correta.

Obviamente, o critério de avaliação acima será aplicado SOMENTE aos trabalhos que possam ser compilados e que não sejam resultado de plágio. Caso o programa não compile receberá a nota 0 (zero). A suspeita de plágio será considerada através de um teste de similaridade de código que envolverá os programas de todos os alunos da disciplina. Os alunos cujos programas sejam "semelhantes" serão convocados para uma entrevista com o professor da disciplina. Nesta entrevista, o aluno será questionado se a similaridade é um mero acaso do destino ou o resultado de plágio. Se o aluno afirmar que a similaridade é um mero acaso do destino, ele será indagado sobre várias partes do código que ele submeteu e a nota do programa dele será resultante das respostas fornecidas para essas perguntas. Caso o aluno afirme que houve plágio, os programas de todos os alunos envolvidos no plágio receberão nota zero.