UNIVERSIDADE FEDERAL DE SANTA CATARINA
INE5412 – Sistemas Operacionais I
Profs. Giovani Gracioli e Patrícia Plentz
2024/01

Simulação de Alocação e Gerenciamento de Memória Livre

O presente trabalho tem por objetivo escrever um programa para simular o comportamento do gerenciamento de memória livre em sistemas operacionais modernos. O programa deve simular o gerenciamento de memória livre com Bitmap e Lista Duplamente Encadeada.

O programa deve receber como parâmetro o método de gerenciamento de memória, sendo 1 para bitmap ou 2 para lista, a quantidade de memória que deve ser gerenciada em bytes, o tamanho de bloco mínimo de alocação em bytes e o algoritmo usado na alocação de memória para procurar um espaço livre. Após isso, o programa deve receber um pedido de alocação ou desalocação de memória. O pedido de alocação tem o seguinte formato:

- A BYTES ID

Onde A indica que é uma alocação, bytes é a quantidade de bytes a serem alocadas e 1 seria um identificador (ID) único de alocação. Para a alocação, cada grupo deverá implementar pelo menos dois algortimos baseados em bin packing para encontrar um espaço de memória vazio que satisfaça a requisição.

O pedido de desalocação tem o seguinte formato:

- D ID

Onde D indica que é uma desalocação e ID associa a área desalocada com o ID de alocação.

A sequência de alocação e/ou desalocação pode ser recebida por linha de comando através de um arquivo de entrada, conforme exemplo abaixo:

```
./simulador < entrada.txt

ou

cat entrada.txt | ./simulador
```

O arquivo entrada.txt tem o seguinte formato (desconsiderar os comentários:

```
1 //aqui indica que o gerenciamento de memória livre é o bitmap
8388608 // tamanho de memória livre de 8 MB
4 //bloco mínimo de alocação de 4 bytes
1 //1 ou 2 - escolhe o algoritmo de alocação
```

```
A 1024 1 //alocação de 1024 bytes com ID 1
A 100 2  //alocação de 100 bytes com ID 2
A 50 3   //alocação de 50 bytes com ID 3
D 2      //libera a área de memória alocada associada ao ID 2
```

Ao final do programa de simulação, deve apresentar o estado do gerenciador de memória. Caso o método de gerenciamento seja listas, a saída deverá seguir o seguinte formato: tamanho do bloco e um indicativo se está livre ou ocupado (1 para ocupado, 0 para livre). O trecho abaixo exemplifica uma saída (desconsiderar os bytes):

```
10 1 //10 bytes ocupados
2 0 //2 bytes livres
1024 1 //1024 bytes ocupados
```

Caso o método de gerenciamento seja bitmap, a saída deverá imprimir o estado final do bitmap, considerando o bit 1 para ocupado e 0 para livre.

```
00011100111111100000000001111111111111111111
```

Atenção: use exatamente esse formato de saída.

Além disso, em ambas as saídas, deve conter um resumo da situação do gerenciador de memória: quantidade de bytes em uso, quantidade de bytes alocados ao todo, quantidade de bytes desalocador, número de requisições de alocação e número de requisições para desalocação de memória. Usem exatamente o exemplo de saída abaixo:

```
10254   //quantidade de bytes em uso (ocupados)
129410 //quantidade de bytes alocados
119156 //quantidade de bytes desalocados
10      //número de alocações
8       //número de desalocações
```

OBS: não deve-se utilizar as estruturas de dados disponíveis nas bibliotecas de linguagem C++, nem bibliotecas prontas para implementação dos algoritmos de bin packing. Cada grupo deverá implementar sua própria solução.

Formato de Entrega e Avaliação

O trabalho em grupo deverá ser entregue no Moodle no dia especificado pela tarefa. Todos os arquivos contendo o código do trabalho, bem como Makefile e um diagrama de classes UML deverão ser submetidos pelo Moodle (compactar um diretório inteiro contendo todos os arquivos – NÃO COMPACTAR ARQUIVOS INDIVIDUAIS FORA DO DIRETÓRIO).

A avaliação se dará em 4 fases:

1. Avaliação de compilação: compilar o código enviado. Caso haja erros de compilação, a nota do trabalho será automaticamente zerada.

2. Avaliação de execução: para validar que a solução executa corretamente sem falhas de segmentação. Caso haja falhas de segmentação, a nota é zerada. Será também avaliado o uso

de variáveis globais (-5 pontos) e vazamentos de memória (-20%). Comparação da saída com valores de testes esperados.

3. Avaliação da organização do código: busca-se nesta fase avaliar a organização do código orientado a objetos e o seguimento das diretrizes do trabalho (saída, uso apropriado de classe, com projeto orientado a objeto, reuso de software, hierarquia de classes, implementação dos algoritmos e estruturas de dados, etc). Deve-se usar classes e objetos e não estilo de programação baseado em procedimentos (como na linguagem C). Alguns itens para avaliação são: (i) funcionamento do programa; (ii) saída do programa (conforme especificação); (iii) clareza do código (utilização de comentários e nomes de variáveis adequadas); (iv) compilação sem warnings; (v) sem vazamento de memória; (vi) reuso de software e hierarquia de classes;

4. A quarta fase consiste na apresentação do trabalho em dia e horário agendado pelo professor. Durante as apresentações, o professor irá avaliar o **conhecimento individual dos alunos sobre os conteúdos teóricos e práticos vistos em aula e sobre a solução adotada no trabalho**. A nota atribuída à cada aluno *i* no trabalho ($NotaTrabalho_i$) será calculada da seguinte forma, onde A_i é a nota referente à apresentação do aluno *i* e *S* é a nota atribuída à solução do trabalho:

$$NotaTrabalho_i = \frac{A_i \, x \, S}{10}$$

Plágio não será tolerado em nenhuma hipótese ao longo dos trabalhos, acarretando em nota 0 a todos os envolvidos.

Referências

- Sistemas Operacionais Modernos. 3ª edição. Andrew S. Tanembaum. Pearson. 2010.