Gerência de Memória

Fabrício R. Pujol[1], Mateus Eckert Reckziegel[1]

1Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

90.619-900 – Porto Alegre – RS – Brasil

{fabricio.pujol, mateus.reckziegel}@acad.pucrs.br

Resumo. *Este trabalho descreve o problema descrito na apresentação do segundo trabalho da disciplina de Sistemas Operacionais e a sua resolução, o qual se refere a implementação de um programa que controla a alocação de memória através da gerência de memória por partições variáveis, para tanto temos este trabalho estruturado de forma que tenhamos uma breve introdução sobre o assunto, a descrição do problema, descrição da solução, resultados apresentados, resultados discutidos e uma conclusão.*

1. Descrição do problema

No problema apresentado pelo professor Avelino Francisco Zorzo foi pedido um código que lesse um arquivo de entrada de dados o qual continha as seguintes informações, modo das alocações, sendo o número 1 considerado com fixo, os limites da memória com o endereço inicial e endereço final, as solicitações de memória através de blocos e as liberações desses blocos, sendo lidos como “S" e “L”, respectivamente. Este programa deve, também informar quando houver fragmentação externa no sistema, demonstrando como a memória está organizada e quais os blocos estão ocupados e quais estão livres.

Vale ressaltar que não é preciso controlar o tempo e as alocações e liberações serão realizadas na ordem que chegaram e puderem ser atendidas. No caso de uma alocação não puder ser atendida, deve ser verificada a situação de ela poder ser atendida no momento em que uma liberação ocorrer. Sendo assim, segue abaixo o exemplo de entrada de dados, conforme explicitados na descrição do problema.

```
Exemplo:
1     // modo fixo - para o modo aleatório este valor será 2
100   // mi
1250  // mf
S 250 // bloco 1
S 100 // bloco 2
S 200 // bloco 3
L 2   // libera bloco 2
S 150 // bloco 4
S 150 // bloco 5
S 150 // bloco 6
L 5 // libera bloco 5
S 200 // neste momento existe fragmentação
```

Figura 1. Exemplo de arquivo de entrada

```
Exemplo de saída quando acontecer fragmentação.
100-350   bloco 1 (tamanho 250)
350-450   livre (tamanho 100)
450-650   bloco 3 (tamanho 200)
650-800   bloco 4 (tamanho 150)
800-950   livre (tamanho 150)
950-1100  bloco 6 (tamanho 150)
1100-1250 livre (tamanho 150)
400 livres, 200 solicitados - fragmentação externa.
```

Figura 2. Exemplo gráfico de saída

Acima temos um exemplo de como deve ser o Feedback do programa, como demonstrado na descrição do problema.

2. Descrição da solução

Para a solução deste problema escolhemos utilizar a linguagem JAVA e para representar os blocos e os seus comportamentos utilizamos a estrutura de dado Lista duplamente encadeada. Uma lista duplamente encadeada é uma estrutura de dados ligada que consiste de um conjunto de registro sequencialmente ligados chamados de nós e é uma extensão da lista simplesmente ligada. Cada Nó tem dois campos chamados de links ou enlaces, que são referências para o nó anterior e posterior. Os links anteriores e posteriores dos nós inicial e final, respectivamente, apontam para algum tipo de terminador, tipicamente um nó sentinela ou nulo, para facilitar o percorrimento da lista.

Já para a alocação dos blocos de memória utilizamos os *First-Fit*, sendo este o algoritmo mais simples de alocação onde o algoritmo procura ao longo da lista de segmentos de memória até encontrar o primeiro espaço livre que caiba a alocação solicitada, quebrando este segmento em duas partes, a primeira para o processo que fez a requisição de alocação e a outras para a memória não usada. É um algoritmo rápido pois finaliza a busca o mais cedo possível.

3. Resultados apresentados

Neste item apresentaremos o arquivo de entrada e as suas respectivas saídas de acordo com cada operação do arquivo de entrada. Sendo assim, segue abaixo o arquivo de entrada chamado “entrada.txt".

```
1 // modo fixo –
100 // mi
1250 // mf
S 250 // bloco 1
S 100 // bloco 2
S 200 // bloco 3
L 2 // libera bloco 2
S 150 // bloco 4
S 150 // bloco 5
S 150 // bloco 6
L 5
S 200 // neste momento existe fragmentação
L 1 // neste momento tem que alocar o bloco 7 que tinha ficado trancado
S 200 // fragmentação novamente
L 6 // neste momento pode atender o bloco 8 que tinha ficado trancado
S 600 // não tem memória suficiente
L 3
L 4
S 700
```

Figura 3. Arquivo de entrada de dados

Devemos considerar que cada linha deste arquivo é uma operação diferente e que para isso teremos um Feedback do programa o qual será explicado agora passo a passo.

```
Memória:
Livre: 100 - 1250; Tamanho: 1151

Operações pendentes:
Nenhuma
```

Figura 4. Primeira saída

```
Alloc memory - 250

Bloco 1: 100 - 349; Tamanho: 250
Livre: 350 - 1250; Tamanho: 901
```

```
Alloc memory - 100

Bloco 1: 100 - 349; Tamanho: 250
Bloco 2: 350 - 449; Tamanho: 100
Livre: 450 - 1250; Tamanho: 801
```

```
Alloc memory - 200

Bloco 1: 100 - 349; Tamanho: 250
Bloco 2: 350 - 449; Tamanho: 100
Bloco 3: 450 - 649; Tamanho: 200
Livre: 650 - 1250; Tamanho: 601
```

Figura 5. Saídas após as primeiras alocações

Aqui podemos observar que as primeiras linhas do arquivo de entrada serviram para delimitar o tamanho da memória inicial, como demonstrado na Figura 4, já na Figura 5, podemos ver que tivemos 3 alocações bem sucedidas representadas pelos blocos 1,2 e 3.

```
Free memory - 2

Bloco 1: 100 - 349; Tamanho: 250
Livre: 350 - 449; Tamanho: 100
Bloco 3: 450 - 649; Tamanho: 200
Livre: 650 - 1250; Tamanho: 601
```

Figura 6. Saídas após a primeira liberação de memória

```
Alloc memory - 150

Bloco 1: 100 - 349; Tamanho: 250
Livre: 350 - 449; Tamanho: 100
Bloco 3: 450 - 649; Tamanho: 200
Bloco 4: 650 - 799; Tamanho: 150
Livre: 800 - 1250; Tamanho: 451
```

```
Alloc memory - 150

Bloco 1: 100 - 349; Tamanho: 250
Livre: 350 - 449; Tamanho: 100
Bloco 3: 450 - 649; Tamanho: 200
Bloco 4: 650 - 799; Tamanho: 150
Bloco 5: 800 - 949; Tamanho: 150
Livre: 950 - 1250; Tamanho: 301
```

```
Alloc memory - 150

Bloco 1: 100 - 349; Tamanho: 250
Livre: 350 - 449; Tamanho: 100
Bloco 3: 450 - 649; Tamanho: 200
Bloco 4: 650 - 799; Tamanho: 150
Bloco 5: 800 - 949; Tamanho: 150
Bloco 6: 950 - 1099; Tamanho: 150
Livre: 1100 - 1250; Tamanho: 151
```

Figura 7. Alocações bem sucedidas

Dando continuidade às instruções do arquivo de entrada temos uma liberação do bloco 2, representado pela Figura 6 e três alocações de memórias representadas pelos blocos 4,5 e 6.

```
Free memory - 5

Bloco 1: 100 - 349; Tamanho: 250
Livre: 350 - 449; Tamanho: 100
Bloco 3: 450 - 649; Tamanho: 200
Bloco 4: 650 - 799; Tamanho: 150
Livre: 800 - 949; Tamanho: 150
Bloco 6: 950 - 1099; Tamanho: 150
Livre: 1100 - 1250; Tamanho: 151
```

```
Alloc memory - 200

Falhou

Houve fragmentação externa. Espaço livre = 401
```

Figura 8. Liberação de memória e falha na alocação

Após liberar mais um bloco, o arquivo nos faz a solicitação de uma alocação de tamanho 200 o que não é possível no momento, sendo assim guardamos esta operação para que ela seja tentada novamente no futuro e haverá fragmentação externa visto que o espaço livre não contíguo é maior que o espaço requisitado.

```
Free memory - 1

Livre: 100 - 449; Tamanho: 350
Bloco 3: 450 - 649; Tamanho: 200
Bloco 4: 650 - 799; Tamanho: 150
Livre: 800 - 949; Tamanho: 150
Bloco 6: 950 - 1099; Tamanho: 150
Livre: 1100 - 1250; Tamanho: 151

Tentando executar operações pendentes...

Alloc memory - 200

Bloco 7: 100 - 299; Tamanho: 200
Livre: 300 - 449; Tamanho: 150
Bloco 3: 450 - 649; Tamanho: 200
Bloco 4: 650 - 799; Tamanho: 150
Livre: 800 - 949; Tamanho: 150
Bloco 6: 950 - 1099; Tamanho: 150
Livre: 1100 - 1250; Tamanho: 151
```

Figura 9. Liberação de bloco e alocação de solicitações pendentes

Podemos observar que após a nova liberação de bloco, através da liberação do bloco 1, poderemos rearranjar a solicitação anterior, abrindo um novo bloco, o bloco 7.

```
Alloc memory - 200

Falhou

Houve fragmentação externa. Espaço livre = 451
```

Figura 10. Falha na alocação de bloco

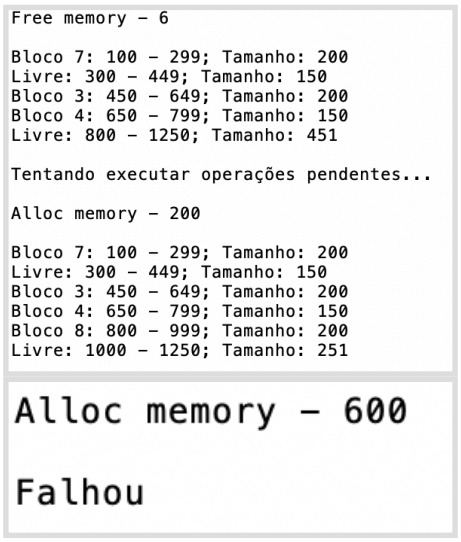

Figura 10. Liberação de bloco e realocação de pendentes

Após falha na realocação do bloco de tamanho 200, o guardamos para uma nova tentativa posterior, na próxima operação liberamos o bloco 6, após isso teremos espaço suficiente para a alocação da solicitação anterior. Depois dessa reorganização dos blocos temos uma solicitação de alocação de tamanho 600, o que não é possível e nem provoca fragmentação externa.

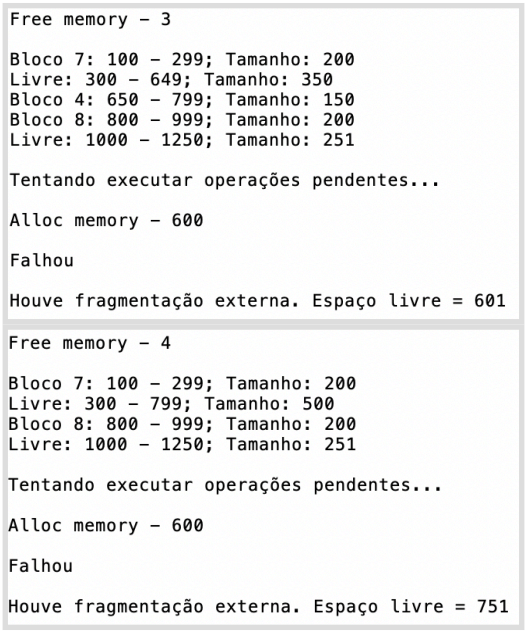

Figura 11. Liberação de blocos e falha na alocação de bloco

```
Alloc memory - 700

Falhou

Houve fragmentação externa. Espaço livre = 751
```

Figura 12. Falha na alocação de bloco

Após duas sucessivas liberações de blocos e consequentemente falhas na alocação da solicitação de tamanho 600, com fragmentação externa após as liberações. Temos uma nova tentativa falha de alocação de tamanho 700 o qual tem também fragmentação externa.

```
Resultado final:

Memória:
Bloco 7: 100 - 299; Tamanho: 200
Livre: 300 - 799; Tamanho: 500
Bloco 8: 800 - 999; Tamanho: 200
Livre: 1000 - 1250; Tamanho: 251

Operações pendentes:
Alloc memory - 600
Alloc memory - 700
```

Figura 13. Resultado Final

Por fim, terminamos de interpretar o arquivo de entrada, com dois blocos ainda alocados, o bloco 7 e 8, com 751 de espaço livre e duas alocações pendentes, uma de 600 e outra de 700.

4. Conclusão

Após a resolução deste problema pudemos atestar a fácil implementação de algoritmo *First-Fit* para alocação de memória visto que seria menos performático se tivéssemos que ficar testado os tamanhos de memória para receber as alocações, conforme os algoritmos de *Best-Fit* e *Worst-Fit*.

5. Referências bibliográficas

Sistemas Operacionais/Gerência de memória. Disponível em https://pt.wikibooks.org/wiki/Sistemas_operacionais/Gerência_de_memória. Último acesso: 25/06/2019.

Gerenciamento de memória. Disponível em ftp://ftp.dca.fee.unicamp.br/pub/docs/ea876/so-cap4.pdf. Último acesso: 25/06/2019.

Lista duplamente ligada. Disponível em https://pt.wikipedia.org/wiki/Lista_duplamente_ligada. Último acesso: 01/05/2019.