MYmalloc: Uma solução personalizada para gerenciamento de memória em programas em C

Hugo V. A. Rezende, Josué B. R. Pacheco e Maxwel Batalha da S. Lopes, *Estudantes, UFRRJ (Universidade Federal Rural do Rio de Janeiro)*

***Resumo* — Este estudo comparou o desempenho da função MYmalloc, que é uma implementação personalizada do malloc, com a função malloc em relação à alocação de memória em estruturas de dados, como listas encadeadas e pilhas. Os resultados mostraram que o MYmalloc é mais rápido do que o malloc em aplicações que alocam e liberam memória repetidamente de mesmo tamanho. Além disso, o ganho percentual na alocação de memória se manteve constante conforme o número de elementos aumentou, indicando que o MYmalloc é eficiente mesmo em estruturas maiores. Em geral, os resultados sugerem que o MYmalloc pode ser uma opção viável para alocação de memória em estruturas de dados menores, mas é importante considerar os trade-offs em relação à fragmentação externa.**

Abstract — This study compared the performance of the MYmalloc function, which is a custom implementation of malloc, with the malloc function in relation to memory allocation in data structures such as linked lists and stacks. The results showed that MYmalloc is faster than malloc in applications that repeatedly allocate and free memory of the same size. Additionally, the percentage gain in memory allocation remained constant as the number of elements increased, indicating that MYmalloc is efficient even in larger structures. Overall, the results suggest that MYmalloc may be a viable option for memory allocation in smaller data structures, but it is important to consider trade-offs regarding external fragmentation.

I. Introdução

O gerenciamento de memória é uma tarefa fundamental na programação de sistemas, sendo responsável por alocar e liberar recursos de memória. O malloc é uma função padrão da linguagem C que permite a alocação dinâmica de memória durante a execução de um programa. No entanto, embora seja uma ferramenta útil, sua eficiência em termos de tempo de execução pode ser melhorada.

Neste trabalho, propomos a criação da função MYmalloc, que tem como objetivo superar o desempenho do malloc padrão em termos de tempo de execução. Para alcançar esse objetivo, elaboramos uma estratégia para gerenciamento de memória.

Este artigo está organizado em cinco seções. Na seção 1, introduzimos o tema geral do nosso trabalho. Na seção 2, apresentamos a motivação específica para o nosso estudo: desenvolver uma melhoria no tempo de execução para a função malloc. Na seção 3, descrevemos a solução proposta usando duas novas funções personalizadas chamadas de MYmalloc e MYfree, assim, discutimos as diferentes estratégias que foram consideradas. Na seção 4, apresentamos os resultados dos testes de desempenho realizados com a MYmalloc em comparação com o malloc padrão. Na seção 5, discutimos as conclusões que podemos tirar desses resultados. Por fim, na seção 6, apresentamos as referências utilizadas neste estudo.

II. Motivação

O gerenciamento eficiente de memória é uma tarefa crítica para o desenvolvimento de programas em C. À medida que os programas crescem em complexidade e tamanho, o gerenciamento de memória pode se tornar um gargalo para o desempenho geral do sistema. A função malloc é amplamente utilizada para alocação dinâmica de memória em programas em C, mas seu desempenho em termos de tempo de execução pode ser afetado por vários fatores, incluindo o tamanho da memória alocada.

Com a crescente demanda por aplicativos de alta performance em várias áreas, há uma necessidade crescente de otimizar o desempenho do gerenciamento de memória. Uma das maneiras de alcançar esse objetivo é criando uma função personalizada para gerenciamento de memória que seja mais eficiente do que o malloc padrão.

A criação do MYmalloc é motivado pela necessidade de melhorar o desempenho do gerenciamento de memória em programas desenvolvidos em C que exigem uma grande quantidade de alocação e desalocação de memória em tamanhos semelhantes. Nesses casos, a MYmalloc pode oferecer uma vantagem significativa em relação ao malloc padrão em termos de tempo de execução e uso eficiente de memória.

III. Solução proposta

A função malloc é implementada por meio de um gerenciador de memória que aloca e libera blocos de memória de tamanho variável. Esse gerenciador mantém uma lista de blocos livres de memória que podem ser usados para atender

às solicitações de alocação de memória. Quando a função malloc é chamada, o gerenciador de memória procura na lista de blocos livres um bloco que seja grande o suficiente para atender à solicitação de alocação de memória. Se um bloco adequado for encontrado, ele é alocado e o ponteiro para o início desse bloco é retornado. Caso contrário, o gerenciador de memória solicita mais memória ao sistema operacional e adiciona um novo bloco livre à lista[2].

Para evitar a fragmentação interna, o gerenciador de memória verifica se o bloco tem tamanho suficiente. Caso o bloco seja maior que o necessário, é realizada uma operação para criar um novo bloco com a memória restante. Para evitar a fragmentação externa, a função free pode combinar blocos adjacentes para formar um bloco maior[3].

No entanto, essas técnicas usadas pelo gerenciador de memória para evitar a fragmentação de memória podem afetar negativamente o desempenho, pois requerem a realização de operações adicionais para combinar e dividir blocos.

Nossa solução para o tempo de alocação do MYmalloc está justamente nas operações realizadas pela função free. Mas antes de abordar a solução, vamos explicar como a função MYmalloc e MYfree são implementadas.

Nós utilizamos uma lista encadeada com os campos “tamanho” e “prox”. O campo tamanho indica o tamanho do bloco de memória, e o campo próximo guarda o endereço de memória do próximo bloco. Quando é alocada memória pela primeira vez, a função MYmalloc solicita ao sistema operacional, através da função sbrk, memória suficiente para alocar um bloco de memória mais a quantidade de memória solicitada no parâmetro da função MYmalloc. A função sbrk retorna o endereço do início do bloco de memória e a função MYmalloc realiza a operação de pular o espaço ocupado pelo bloco e retorna o endereço de memória livre. Quando já foram alocadas e liberadas memórias antes, a lista de blocos livres já contém blocos prontos, e o MYmalloc não precisará chamar a função sbrk. Ele percorre a lista de blocos livres e utiliza a política first-fit para encontrar um bloco para alocar memória. Quando o bloco é encontrado, ele verifica se o bloco é maior que o necessário. Quando o bloco é maior que o necessário, ele cria um novo bloco com o que sobrou da memória, coloca o bloco na lista de blocos livres e realiza uma operação do no endereço do bloco para retornar o endereço da memória livre.

A função MYfree recebe como parâmetro a variável que guarda o endereço de memória, e ela realiza a operação para chegar ao endereço do bloco que gerencia esse espaço de memória. A função MYfree apenas coloca o bloco na lista de blocos livres para poder ser acessado novamente.

Diferente da função free, a MYfree não combina blocos adjacentes. Os blocos são apenas colocados na lista de blocos livres para serem acessados novamente. Implementamos essa função pensando em aplicações que alocam e liberam a mesma quantidade de memória várias vezes. Como o MYfree não realiza a operação de combinar blocos, o MYmalloc não precisa dividir blocos de memória, já que eles já estão na lista com o tamanho necessário. Além disso, para combinar os blocos na lista de blocos livres, a função free precisa ordenar os blocos de acordo com seus endereços e como o MYfree não realiza essa operação, apenas adicionar o bloco na lista já é suficiente, ganhando mais tempo na liberação de memória.

Com essa abordagem, esperamos que a nossa solução possa oferecer um desempenho mais rápido e eficiente em relação ao malloc padrão, especialmente em cenários de alocação e desalocação frequentes de memória do mesmo tamanho. Além disso, a nossa solução pode ajudar a simplificar o gerenciamento de memória e reduzir o tempo de execução do programa em geral. No entanto, é importante observar que quando alocamos e liberamos quantidades de memória diferentes utilizando o MYmalloc e o MYfree, pode ocorrer muita fragmentação externa, o que gera um desperdício de memória e, consequentemente, mais chamadas de sistema para alocar memória. Isso pode acabar tornando o MYmalloc e o MYfree piores em tempo de alocação e liberação do que o malloc padrão.

IV. Avaliação

Foram selecionados a lista encadeada e a pilha como estruturas para testes, pois são amplamente utilizadas e possuem características favoráveis à técnica de melhoria escolhida. As duas estruturas alocam e liberam a mesma quantidade de memória, já que seus nós e elementos são sempre do mesmo tamanho.

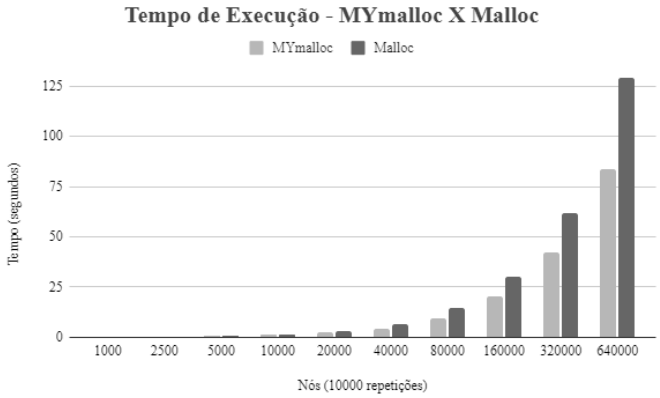
Para garantir a precisão dos nossos testes, utilizamos um computador com as seguintes especificações: processador Intel(R) Core(TM) i3-10105F CPU @ 3.70GHz 3.70 GHz, 16GB de memória RAM e sistema operacional Windows 10.

Os testes foram realizados com as mesmas estruturas de laços e nas mesmas quantidades de repetições para garantir que as comparações fossem justas e precisas. Dessa forma, pudemos avaliar o desempenho das diferentes implementações com base em suas características e técnicas utilizadas, sem que houvesse interferência de outros fatores externos. Além disso, utilizamos a biblioteca nativa do C, time.h, para medir o tempo de execução de cada implementação.

Para garantir resultados confiáveis, cada implementação com suas variações de tamanho foi executada dez vezes, e uma média aritmética foi calculada a partir dos resultados. Dessa forma, obtivemos valores mais precisos e consistentes para avaliar o desempenho das estruturas testadas.

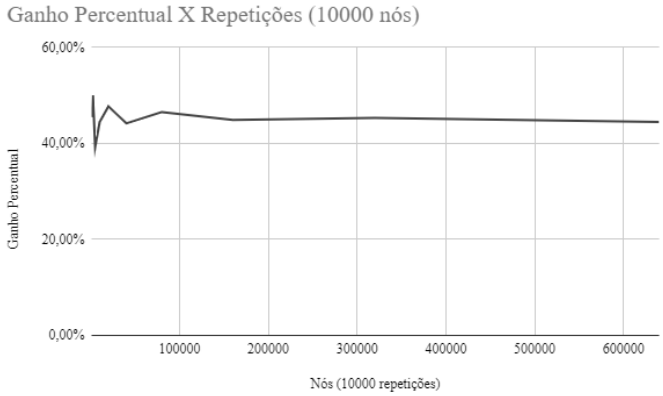
Teste Lista Encadeada				
	Segundos			
Nós (10000 repetições)	**MYmalloc**	**Malloc**	**Diferença**	**Ganho percentual**
1000	0,1	0,14	0,04	40,00%
2500	0,26	0,39	0,13	50,00%
5000	0,53	0,77	0,24	45,28%
10000	1,08	1,6	0,52	48,15%
20000	2,2	3,19	0,99	45,00%
40000	4,43	6,56	2,1	47,40%
80000	9,45	14,61	5,16	54,60%
160000	20,37	30,31	9,94	48,80%
320000	42,43	61,79	19,36	45,63%
640000	83,65	129,37	45,72	54,66%

(figura 1 - tabela de testes da lista encadeada)



(figura 2 - Gráfico comparativo da lista encadeada)

Os testes realizados com a estrutura de dados lista encadeada foram executados com 10000 repetições e com variação no número de elementos, como mostrado nas figuras 1 e 2. Os testes realizados foram executados com os mesmos laços de repetição e nas mesmas quantidades para ambas as técnicas utilizadas de alocação de memória: MYmalloc e Malloc. Os resultados mostram que, em todas as variações de tamanho de nó testadas, a técnica de alocação de memória MYmalloc foi mais eficiente que a técnica de alocação utilizada no Malloc, apresentando ganhos percentuais que variaram de 40% a 54,6%. Por exemplo, para 160.000 nós, o tempo de execução usando MYmalloc foi de 20,37 segundos, enquanto que com Malloc foi de 30,31 segundos, uma diferença de 9,94 segundos, ou um ganho percentual de 48,80%. Dessa forma, podemos concluir que a técnica de alocação de memória implementada na função MYmalloc é uma opção vantajosa para otimizar o desempenho de estruturas de dados como Lista Encadeada.

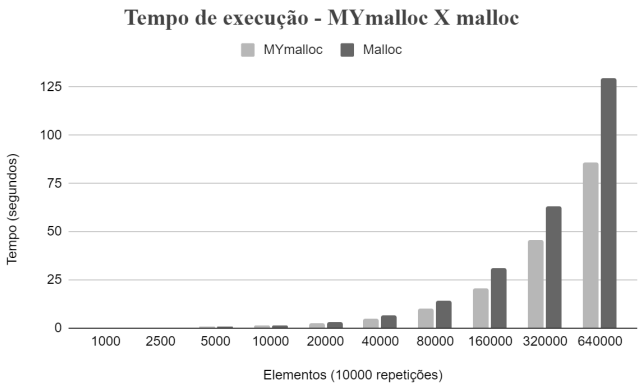


(figura 3 - Gráfico de ganho percentual da lista encadeada)

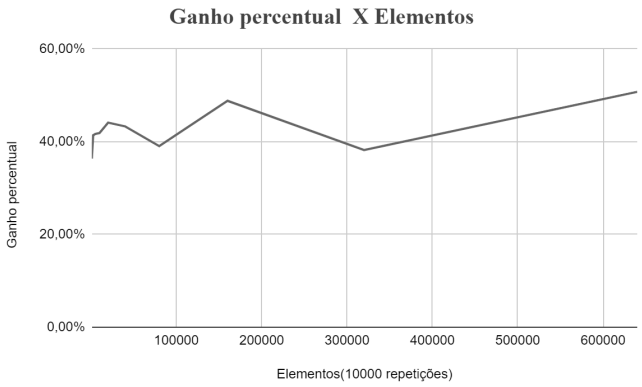
Compreender a linearidade dos ganhos percentuais em relação ao aumento do número de nós da Lista Encadeada é um ponto importante a ser considerado. Como indicado na Figura 3, é possível observar que o ganho percentual se mantém praticamente constante à medida que a quantidade de nós aumenta. Esse comportamento sugere que a técnica utilizada para melhorar a alocação de memória com a função MYmalloc é eficiente mesmo em estruturas de dados maiores. Essa informação é relevante, pois indica que a utilização dessa técnica pode ser benéfica não apenas em estruturas de menor porte, mas também em estruturas de dados mais complexas e com maior demanda de memória.

Teste da Pilha				
	Segundos			
Elementos(10000 repetições)	**MYmalloc**	**Malloc**	**Diferença**	**Ganho percentual**
1000	0,11	0,15	0,04	36,36%
2500	0,29	0,41	0,12	41,38%
5000	0,60	0,85	0,25	41,67%
10000	1,17	1,66	0,49	41,88%
20000	2,38	3,43	1,05	44,12%
40000	4,85	6,95	2,10	43,30%
80000	10,22	14,21	3,99	39,04%
160000	20,75	30,88	10,13	48,82%
320000	45,75	63,23	17,48	38,21%
640000	86,07	129,75	43,68	50,75%

(figura 4 - tabela de testes da pilha)



(figura 5 - Gráfico comparativo da pilha)



(figura 6 - Gráfico de ganho percentual da pilha)

Na estrutura de dados Pilha, os testes foram realizados com as mesmas estruturas de laços e nas mesmas quantidades para ambas as técnicas, com 10000 repetições.

Assim como na Lista Encadeada, os resultados mostram que a técnica de alocação de memória MYmalloc foi mais eficiente do que a técnica utilizada no Malloc em todas as variações de tamanho da Pilha. Os ganhos percentuais variaram de 36,36% a 50,75%, o que indica que a técnica implementada na função MYmalloc é vantajosa para otimizar o desempenho de estruturas de dados como a pilha. Os dados podem ser observados nas figuras 4, 5 e 6.

V. Conclusão

A implementação da alocação de memória customizada utilizando a função MYmalloc apresentou desempenho superior em relação à função malloc padrão nas estruturas de dados Lista Encadeada e Pilha.

Os resultados obtidos demonstraram que a técnica de alocação de memória utilizada na função MYmalloc foi eficiente mesmo em estruturas de dados maiores. Além disso, foi possível observar a linearidade dos ganhos percentuais em relação ao aumento do número de elementos nas Listas Encadeadas e Pilhas, sugerindo que a técnica foi aplicável em diferentes tamanhos de dados.

Em suma, a implementação da função MYmalloc pode ser uma alternativa interessante para melhorar a performance de aplicações que utilizam alocação dinâmica de memória, especialmente em estruturas de dados que sempre alocam e liberam quantidades de memória iguais.

VI. Referências

[1] Strawberryhacker, "Allocator," GitHub, 2021. [Online]. Disponível: https://github.com/strawberryhacker/allocator/blob/master/allocator.c. [Acessado: 20/02/2023].

[2] Free Software Foundation, "The GNU C Library Reference Manual," 2021.

[3] B. W. Kernighan and D. M. Ritchie, "The C Programming Language," Second Edition, Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 1988.