**UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO**
FLÁVIO WILDNER
GABRIEL DELGADO

# Atividade Acadêmica de Sistemas Operacionais

Implementação do “MyMalloc”

# 1.Introdução

Quando um array em um programa C, deve-se informar quantos elementos devem ser reservados para este array. Se você conhece este número a priori, está tudo bem.

Porém, o tamanho de um array pode ser desconhecido por uma série de fatores. Por exemplo, se deseja-se ler todas as linhas de um arquivo e armazená-los em um array de um programa, o tamanho de memória necessário dependerá do tamanho do arquivo. E se este tamanho variar muito de um arquivo para outro, será preciso reservar espaço suficiente para acomodar o maior tamanho de arquivo possível, o que é um desperdício se o programa vai lidar com muitos arquivos de tamanho reduzido e apenas alguns arquivos com tamanho grande.

Se o programa aloca espaço desnecessariamente, isto quer dizer que menos processos poderão ocupar a memória para serem executados. Portanto, durante a gerência do processo de sua tarefa pelo sistema operacional, a transferência do programa do disco para a memória e vice-versa (processo conhecido como swapping), irá demorar mais.

Para permitir que o espaço para estruturas de dados de um programa possam ser alocados durante a execução do mesmo é que a linguagem C define funções de alocação dinâmica de memória, como o malloc.

Porém, o malloc não é a melhor solução para todos os casos. Existem maneiras diferentes de alocar memória dinamicamente que podem buscar ganhos de diferentes formas, como aumentar a velocidade de alocação, diminuir a fragmentação ou até mesmo manter o máximo de coalescência.

Neste trabalho, foi implementado uma função de alocação dinâmica de memória que visa melhor diferentes aspectos abordados pelo malloc padrão do C.

# 2. MyMalloc

Após vistos os problemas do malloc, foi decidido que o plano de otimização abordado no MyMalloc seria em obter ganho em tempo de alocação de memória, para isso, foi usado uma estrutura de dado auxiliar para mapear os blocos de memória livre: uma lista encadeada e o algorítimo de worst fit.

Como o worst fit insere sempre na “pior” posição em que o bloco se encaixa, fazendo restar sempre a maior quantidade de espaço contíguo possível, a fragmentação interna diminui, e através do uso de uma lista encadeada ordenada de forma não crescente por tamanho de bloco livre, ao inserir um novo dado dentro da memória, a primeira posição encontrada sempre será a melhor posição para o worst fit, tornando a alocação de um novo bloco O(1).

Como vantagem, a velocidade de alocação se torna bem mais rápida que o malloc normal, pois não é necessário percorrer a lista de blocos para inserir. Um exemplo seria a alocação de matrizes, em que cada linha dela, consiste em uma alocação separada de um vetor de dados, a chamada repetida desta alocação pode sofrer um speed up através do uso do MyMalloc.

Como desvantagem, é possível observar que é necessário manter a lista de blocos livres sempre ordenada de maneira não crescente para que a alocação se mantenha O(1), então, toda vez que uma operação de free (liberar um bloco da memória) for feita, é preciso garantir que os blocos estejam ordenados, o que pode causar custo extra dependendo do quão fragmentada a memória esteja.

# 3.Resultados

**Primeiro Teste**: Primeiro, é alocada uma matriz, com cada linha com o mesmo tamanho (em bytes), logo em seguida, a cada duas linhas, uma é liberada da memória, e em seguida, essas linhas são realocadas com o mesmo tamanho da linha original. O tempo foi medido do inicio do programa (alocação da primeira matriz) até a realocação das linhas. Abaixo segue os resultados obtidos para o Malloc original e o MyMMalloc em instâncias de diferentes tamanhos:

| **Primeiro Teste: MyMalloc** | **Primeiro Teste: malloc** |
|---|---|
| 100MB - 200ms | 100MB - 240ms |
| 200MB - 380ms | 200MB - 460ms |
| 500MB - 940ms | 500MB - 1130ms |
| 1GB - 1930ms | 1GB - 2240ms |

**Segundo Teste:** Primeiro, é alocada uma matriz, com cada linha com o mesmo tamanho (em bytes), logo em seguida, a cada duas linhas, uma é liberada da memória, e em seguida, essas linhas são realocadas com o tamanhos diferentes, ou seja, não será possível realocar na mesmo posição anterior. O tempo foi medido do inicio do programa (alocação da primeira matriz) até a realocação das linhas. Abaixo segue os resultados obtidos para o Malloc original e o MyMalloc em instâncias de diferentes tamanhos:

| Segundo Teste: MyMalloc | Segundo Teste: malloc |
|---|---|
| 100MB - 205ms | 100MB - 325ms |
| 200MB - 400ms | 200MB - 655ms |
| 500MB - 990ms | 500MB - 1580ms |

**Terceiro Teste**: É alocada uma matriz, com cada linha com o mesmo tamanho (em bytes), logo em seguida, é desalocada todas as linhas. O tempo foi medido do inicio do programa (alocação da primeira matriz) até a realocação das linhas. Abaixo segue os resultados obtidos para o Malloc original e o MyMalloc em instâncias de diferentes tamanhos:

| Terceiro Teste: MyMalloc | Terceiro Teste: Malloc |
|---|---|
| 100MB - 62ms | 100MB - 42ms |
| 200MB - 125ms | 200MB - 86ms |
| 500MB - 310ms | 500MB - 210ms |
| 1GB - 630ms | 1GB - 410ms |

## 4.Conclusão

Através dos testes foi possível observar que a velocidade de alocação do MyMalloc teve um ganho de 60% em relação ao malloc tradicional no melhor caso, enquanto o tempo de desalocação houve uma piora de 53% no pior caso.

Concluímos que o MyMalloc é recomendado a se usar em aplicações em que é necessário a menor latência possível para alocar um conjunto de dados.