# MAC0422 - Sistemas Operacionais

## EP3 - Simulação de Algoritmos de Alocação de Memória

Nome: Matheus Silveira Feitosa

# Considerações Gerais da Implementação

O problema consistiu em simular 4 algoritmos de alocação de memória, sendo eles o First Fit, Next Fit, Best Fit e Worst Fit, de maneira a ressaltar seu funcionamento e suas características, para cada conjunto de operações na memória.

Para isso, foi proposto utilizar arquivos PGM, com valores preenchidos com 0 para unidades ocupadas, e 255 para unidades livres, e nestes, realizar toda a manipulação da memória diretamente.

Desta forma, a simulação foi feita seguindo um fluxo básico de leitura de comando e, em seguida, dada a escolha de algoritmo do usuário, o comando lido era passado para o algoritmo que retornava se a alocação foi feita com sucesso ou não, e em caso negativo, informamos ao usuário qual comando falhou, e incrementamos o contador de falhas de alocação, este que é informado ao final da simulação.

# Variáveis em memória

**De forma global, o único algoritmo que precisou de uma variável de memória entre execuções, foi o Next Fit, que guarda sempre qual é a próxima posição livre.**

**Para os outros algoritmos, as únicas variáveis usadas eram locais, e serviam para demarcar em qual posição devemos começar e finalizar a escrita em uma dada iteração, após identificar um bloco suficientemente grande, tais variáveis não são guardadas após o fim da iteração.**

**Para o Best e Worst Fit, uma outra constante lida do PGM, era usada na inicialização para indicar qual era o tamanho máximo e mínimo, que um dado bloco poderia ter.**

**O principal método de controle de leitura e escrita do PGM, foi baseado no próprio descritor de arquivo, que sempre guardava qual foi a última posição lida, sendo assim, com exceção do Next Fit, a cada iteração, reiniciávamos o descritor para apontar sempre para a primeira posição de memória do PGM.**

# Metodologia de testes

**Buscando enaltecer as características principais de cada algoritmo, em especial a quantidade de falhas e também sua velocidade de execução, criou-se 4 conjuntos de comandos com as características únicas, dadas abaixo:**

- **First Fit: Aproveitando os blocos pequenos no início, tentamos alocar neles 1 unidade de forma alternada com blocos maiores, de forma a dedicar o começo para os blocos pequenos e o final para blocos grandes, diminuindo assim, o consumo de blocos grandes de forma desnecessária.**
- **Next Fit: O foco foi dado em sua velocidade, sendo assim, iniciamos compactando a memória, e em seguida, preenchemos todos os espaços vazios.**
- **Best Fit: a ideia foi de fazer uma alocação pseudo-aleatória, onde começamos fazendo diversas alocações pequenas, e em seguida, intercalamos alocações grandes e pequenas, com o objetivo de preencher os menores espaços possíveis e deixar os espaços grandes, justamente para os pedidos maiores.**
- **Worst Fit: Neste caso a ideia foi justamente oposta ao Best Fit. Começamos alocando de forma pseudo-aleatória, blocos grandes mesclados eventualmente com blocos pequenos, objetivando sempre consumir os maiores blocos possíveis, de forma a não deixar pequenos fragmentos inutilizáveis na memória.**

# Máquinas de teste

**De forma a justificar e padronizar alguns dos resultados obtidos, abaixo estão dispostos as especificações da máquina em que foram executados os testes.**

| Especificações da máquina de testes | |
|---|---|
| Processador | Processador 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz, 3110 Mhz. |
| Número de Processadores | 4 Núcleos, 8 Processadores Lógicos |
| Sistema Operacional | Debian GNU/Linux 12 (bookworm)<br>5.15.167.4-microsoft-standard-WSL2 #1 |
| Memória RAM | Kingston Fury Impact, 8GB, 3200MHz |
| Armazenamento | SSD NVMe KINGSTON SNV2S1000G |

# Resultados dos testes

**Para verificar os resultados obtidos cada simulação foi executada 30 vezes e, em cada uma, foi medido o tempo para cada conjunto de comandos aplicado a cada algoritmo, após cada execução registramos a média e o intervalo de confiança destes experimentos.**

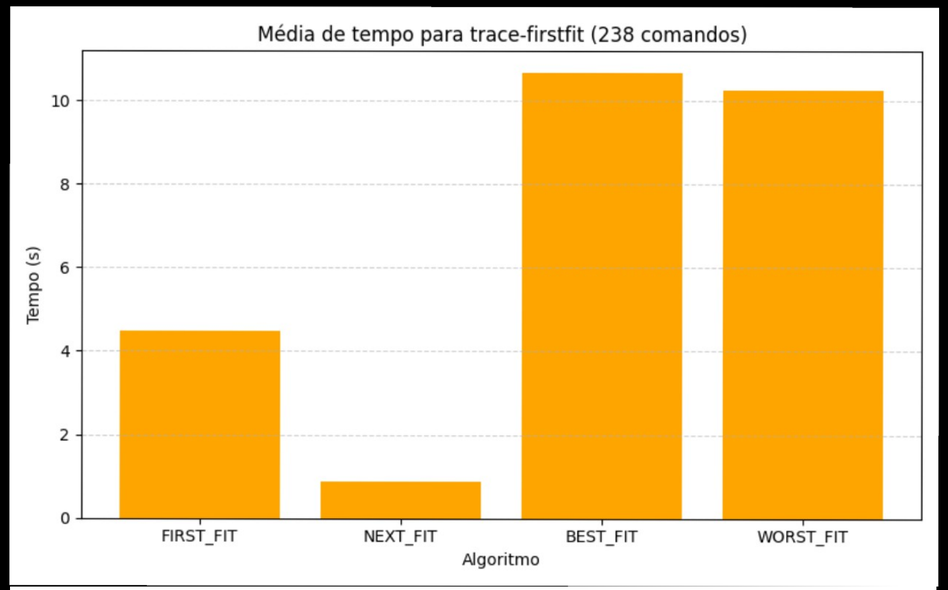
| Velocidade de execução de cada algoritmo em segundos | | | | |
|---|---|---|---|---|
| Arquivo de trace | Algoritmo de escalonamento | | | |
| | FIRST FIT | NEXT FIT | BEST FIT | WORST FIT |
| trace-firstfit (238 comandos) | 4.476646 ±0.364636 | 0.879151 ±0.016298 | 10.663893 ±0.491286 | 10.258399 ±0.444481 |
| trace-nextfit (165 comandos) | 5.619207 ±0.391687 | 0.520479 ±0.103102 | 7.428463 ±0.422370 | 7.359592 ±0.398734 |
| trace-bestfit (930 comandos) | 13.111095 ±0.486453 | 0.812451 ±0.065804 | 39.840009 ±0.540359 | 39.643872 ±0.478225 |
| trace-worstfit (323 comandos) | 6.869281 ±0.407333 | 1.206998 ±0.022303 | 13.764291 ±0.493683 | 13.926024 ±0.474191 |

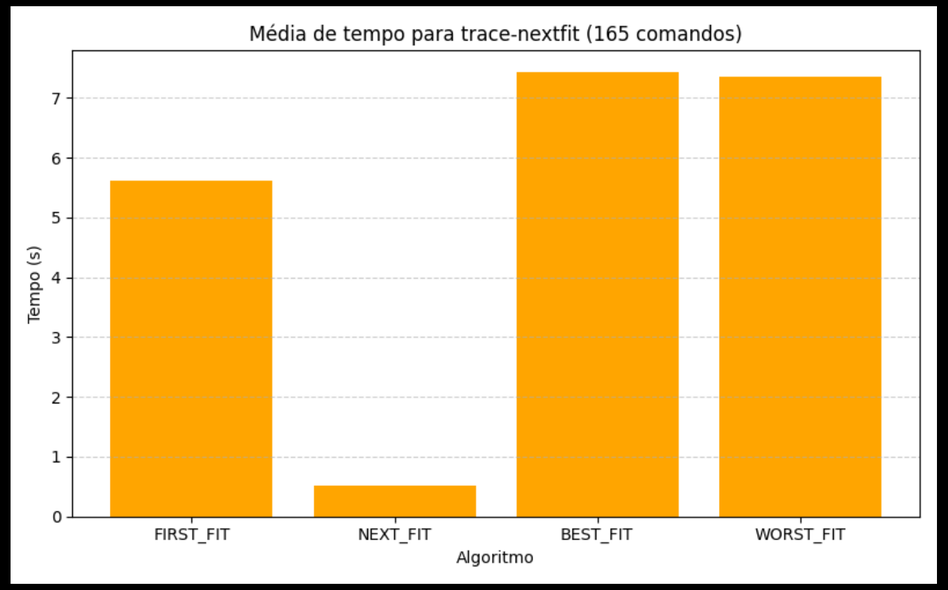
# Resultados dos testes

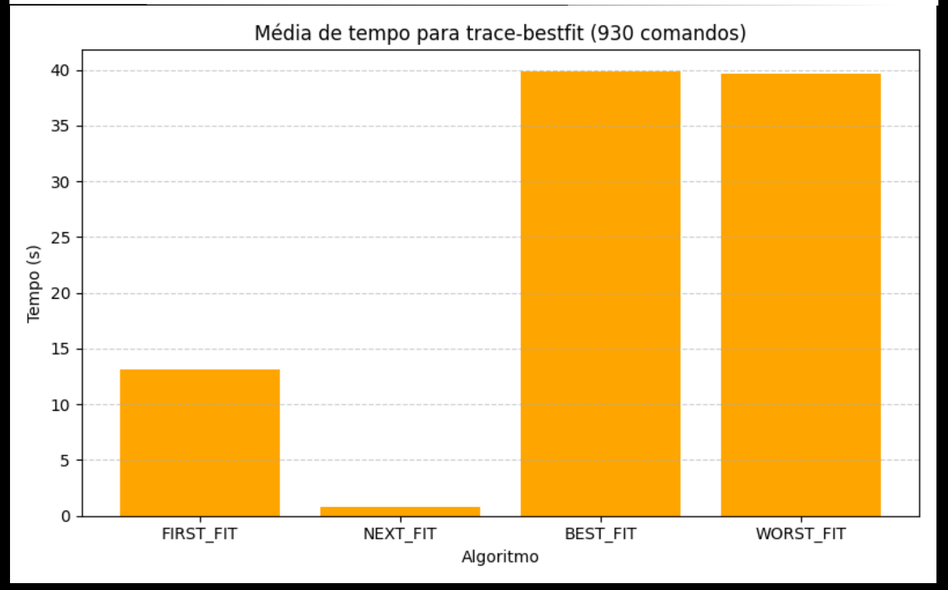
**Afim de verificar também como cada conjunto de comandos afeta a ocorrência de erros, e seu determinismo, para as mesmas 30 execuções, registramos também a quantidade de falhas que aconteceu em cada uma das execuções.**

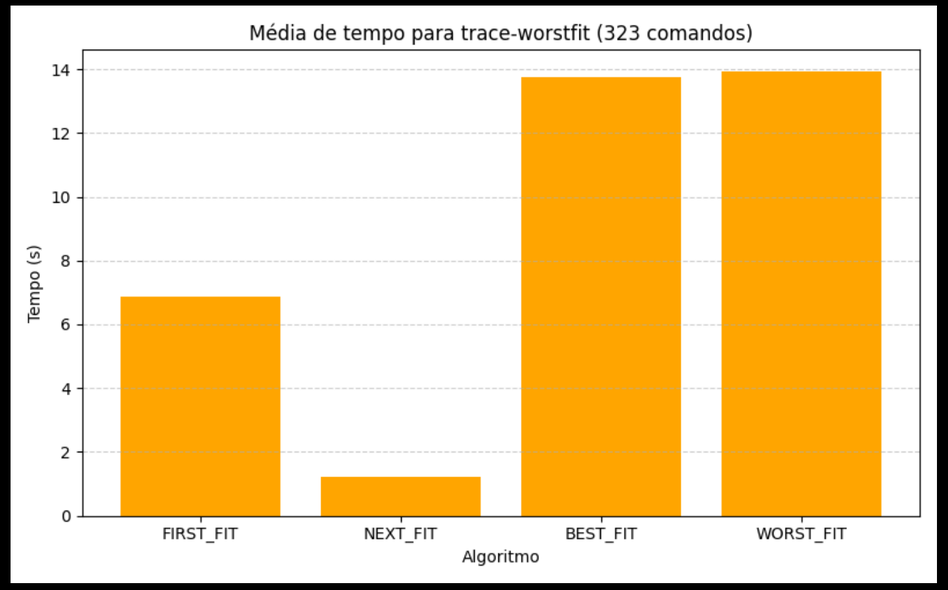
| Número de falhas de alocação para cada algoritmo | | | | |
|---|---|---|---|---|
| Arquivo de trace | Algoritmo de escalonamento | | | |
| | FIRST FIT | NEXT FIT | BEST FIT | WORST FIT |
| trace-firstfit (238 comandos) | **0** | 5 | 2 | 11 |
| trace-nextfit (165 comandos) | 0 | 0 | 0 | 0 |
| trace-bestfit (930 comandos) | 2 | 3 | 0 | 12 |
| trace-worstfit (323 comandos) | 5 | 7 | 5 | 0 |
| Total | 7 | 15 | 7 | 23 |

# Resultados - Tempo de execução

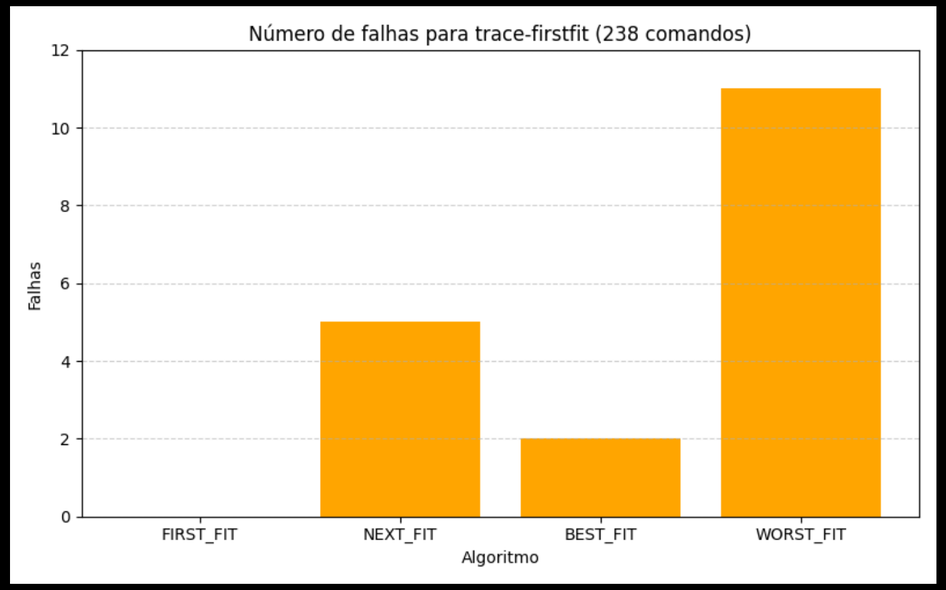


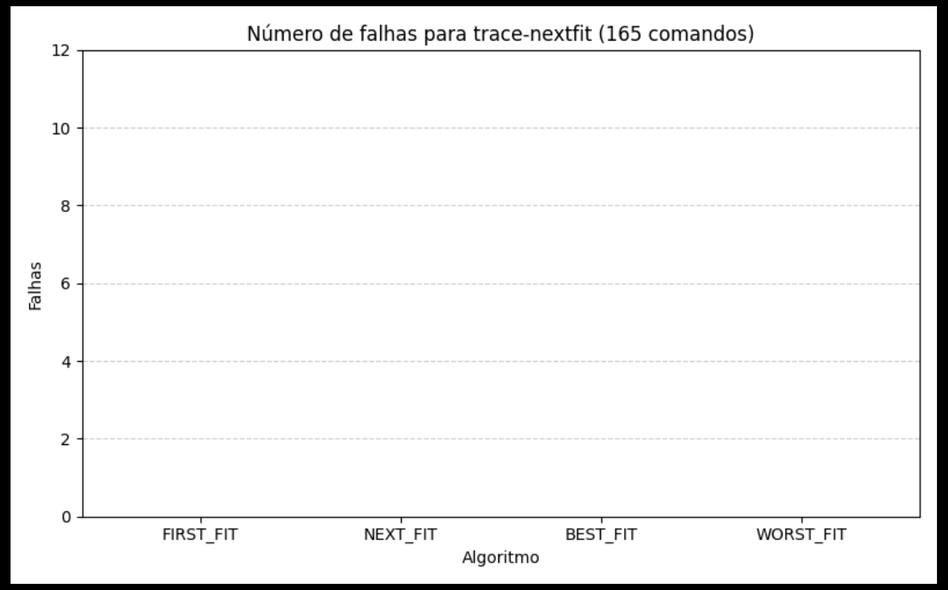



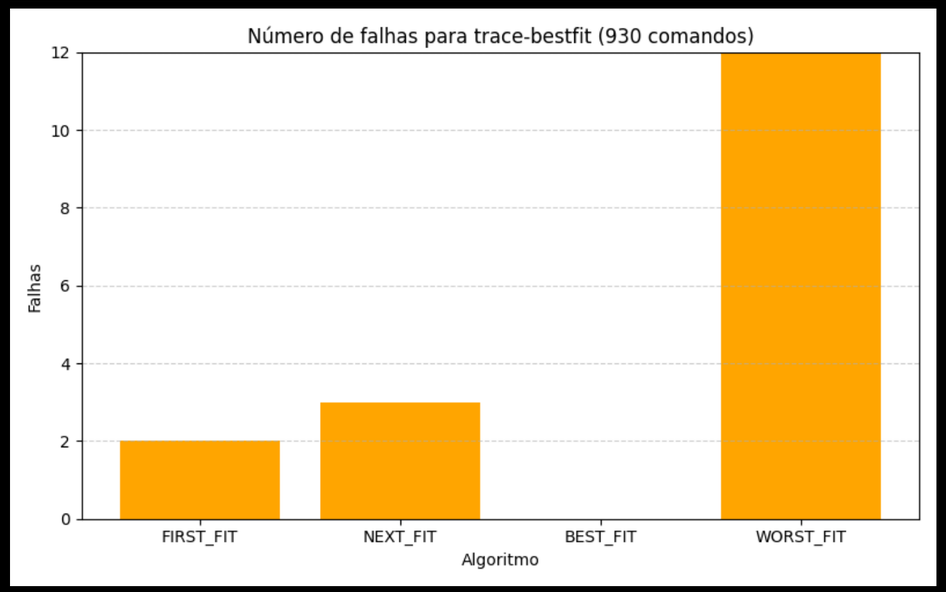



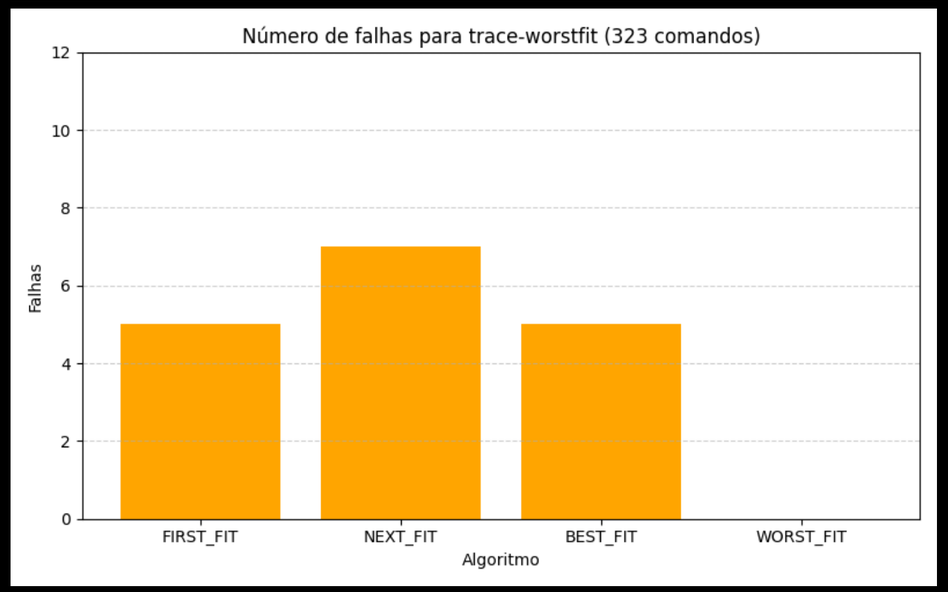



# Resultados - Quantidade de falhas



# Análise dos Resultados

## Tempo de execução.

De maneira geral, os tempos de execução se comportaram exatamente como o esperado, com destaque para o fator extra observado, que foi a dependência da velocidade de execução, com o número de comandos a serem executados, e também como tal fator era sentido por cada um dos algoritmos, assim como detalhado abaixo:

- Começando pelo First Fit, dois pontos principais foram observados, primeiro que, em seu arquivo de trace, mesmo tendo mais comandos do que o trace-nextfit, este ainda teve uma execução mais rápida, o que pode ser justificado pelo fato de que, o conjunto de comandos de trace-firstfit, fazia com que mais da metade das alocações, fossem feitas logo no começo, fazendo com que a posição adequada, fosse rapidamente encontrada, por outro lado, o trace-nextfit por compactar a memória, fazia com que, a cada alocação, o algoritmo fosse cada vez mais longe na memória, consumindo mais tempo. Para os demais, devido a aleatoriedade das alocações, podemos associar seu consumo de tempo, ao número de comandos.

- Para o Next Fit, em praticamente todos os testes,  foi o mais rápido, correspondendo ao esperado, dado que o ponteiro para a próxima posição de alocação, está sempre próxima a um espaço vazio, e ainda, foi notado com seu arquivo de trace que, caso a memória esteja compactada, isto é, haja um espaço contiguo considerável, este algoritmo será o mais eficiente.

- Com respeito ao Best Fit, e Worst Fit, alguns fatores forem observados, primeiramente, que o tempo de execução de ambos, é intrinsecamente dependente do número de comandos, uma vez que este define quantas vezes iremos consultar a memória inteira. Outro fator, que também se relaciona com a quantidade de falhas, é a influência que o ato de registrar na memória tem sobre o tempo, note que, em geral, o Worst Fit tinha tempos de execução menores que o Best Fit, entretanto, ao usarmos o trace-worstfit, o Best Fit executou mais rapidamente que o Worst Fit, este que fez mais alocações, além disso, note que para o caso onde ambos fizeram o mesmo número de registros, ambos tiveram tempos de execução similares.

# Análise dos Resultados

## Quantidade de falhas de alocação

**Inicialmente, era esperado que tanto o Best Fit quanto o Worst Fit, apresentassem uma quantidade similar de falhas de alocações no total, entretanto, tal expectativa não foi atendida, por outro lado, os demais resultados, como por exemplo, o número de falhas individuais de cada algoritmo aplicado a um dado conjunto de comandos, atendeu exatamente as expectativas, assim como elucidado na análise individual de cada um, feita abaixo:**

- **Começando pelo First Fit, em especial para os traces pensados para o Worst Fit e para o Best Fit, vimos que, para o primeiro caso, dado que a parte inicial da memória é composta majoritariamente por blocos pequenos, enquanto os blocos posteriores costumam ser maiores, era esperado que neste algoritmo, os blocos escolhidos tivessem características similares aos do método Best Fit, uma vez que na parte inicial, ficam os menores blocos que, para os valores pedidos, eram suficientes para alocar dado tamanho, mas não eram os maiores de todos, assim gerando também, pequenos fragmentos inutilizáveis de memória, como no caso do Best Fit. Com base nesta ideia, era esperado também que, os resultados obtidos com este algoritmo, fossem similares ao se usássemos o trace-bestfit com o próprio Best Fit, assim como observado.**

- **Indo para o Best fit, o mesmo argumento continua válido, observando inclusive que, ambos tiveram a mesma quantidade de falhas de alocação para o mesmo arquivo de trace, e também a mesma quantidade de falhas no total.**

- **Comparando os resultados obtidos do Worst Fit, vimos que este apresentou o maior número de falhas, entretanto, observando que estas falhas ocorreram, majoritariamente, nos traces de first fit e best fit, temos como justificado tal número de falhas, uma vez que ambos executam uma quantidade considerável de pedidos de tamanho pequeno, fazendo com que o algoritmo Worst Fit, consuma um espaço que poderia ser usado pelos pedidos maiores, sendo, tal observação, condizente com a teoria de tal algoritmo.**

# Análise dos Resultados

## Considerações sobre o Next Fit

- **Por fim, analisando os resultados do Next Fit, vimos que para todos os arquivos de trace utilizados (com exceção do seu próprio arquivo de trace), este algoritmo apresentava sempre alguns erros de alocação, e além disso, tomando a média dos erros totais de todos os algoritmos, vemos que este apresenta o valor mais próximo a média total de erros, indicando que para situações onde há uma fragmentação da memória e não há um padrão na ordenação dos pedidos de alocação, isto é, fazemos os pedidos de forma aleatória, mesmo que com padrões pré-definidos de distribuição de tamanho, temos que este não se beneficia diretamente nem de alocações pequenas nem de alocações grandes em termos de número de falhas. Nos permitindo concluir que seu principal ponto forte está justamente na redução do overhead no tempo de alocação.**