Relatório Técnico: Simulação de Alocação de Memória

Universidade Federal do Ceará (UFC) - Sistemas Operacionais

Equipe: Antônio Gabriel, Erik Bayerlein, Julia Naomi

---

1. Introdução e Objetivo

Este relatório descreve o desenvolvimento de um simulador de gerência de memória focado em estratégias de alocação dinâmica. O objetivo é comparar o desempenho e a fragmentação gerada pelos algoritmos **First Fit**, **Best Fit** e **Worst Fit**, observando como cada política impacta o aproveitamento do espaço físico e a organização de blocos de memória.

Divisão:

Gabriel: Configuração inicial do projeto, base do código, criação de issues, comando de gestão de ciclo de vida.

Erik: Implementação dos Algoritmos.

Julia: Processo de alocação e liberação, comandos de visualização, docs.

2. Decisões de Implementação e Arquitetura

O projeto foi construído em Java, utilizando uma arquitetura modular para separar a lógica de comando, validação e as estratégias de alocação.

2.1 Estruturas de Dados Centrais

- **Memória Física (ArrayList<MemoryBlock>)**: A memória é representada como uma lista de slots discretos (bytes). Cada MemoryBlock armazena seu estado (livre/ocupado) e o ID do processo alocado.
- **Mapa de Alocações (HashMap<Integer, Integer>)**: Armazena o mapeamento ID -> Índice de Início. Esta decisão visa centralizar o controle de onde cada processo começa, evitando buscas exaustivas por IDs em toda a memória durante operações de consulta simples.
- **AllocationInfo (Record/DTO)**: Utilizado para o transporte de dados estruturados entre a lógica de gerência e a interface de saída, garantindo imutabilidade nos relatórios de estatísticas.

2.2 Padrões de Projeto Aplicados

- **Strategy**: Utilizado para encapsular os algoritmos de alocação. A interface IAlgorithmStrategy permite que o simulador alterne entre First, Best e Worst Fit de forma transparente.

- **Command Pattern**: Cada operação do shell (alloc, freeid, show, etc.) foi implementada como um objeto de comando, facilitando a expansão da CLI e a manutenção do código.
- **Factory**: Implementada na classe AlgorithmFactory para instanciar a estratégia correta com base na entrada do usuário.

3. Algoritmos de Alocação

O simulador implementa três políticas distintas para encontrar lacunas (*holes*) na memória:

1. **First Fit**: Aloca o processo no primeiro bloco livre com tamanho suficiente. É o mais rápido, mas pode concentrar fragmentação no início da memória.
2. **Best Fit**: Percorre toda a memória para encontrar a menor lacuna que comporte o processo. Visa minimizar o desperdício imediato, mas pode gerar fragmentos minúsculos e inúteis (fragmentação externa).
3. **Worst Fit**: Aloca o processo na maior lacuna disponível. A estratégia é deixar lacunas remanescentes grandes o suficiente para serem utilizadas por processos futuros.

4. Análise de Fragmentação e Métricas

O sistema calcula métricas vitais para a avaliação da gerência de memória:

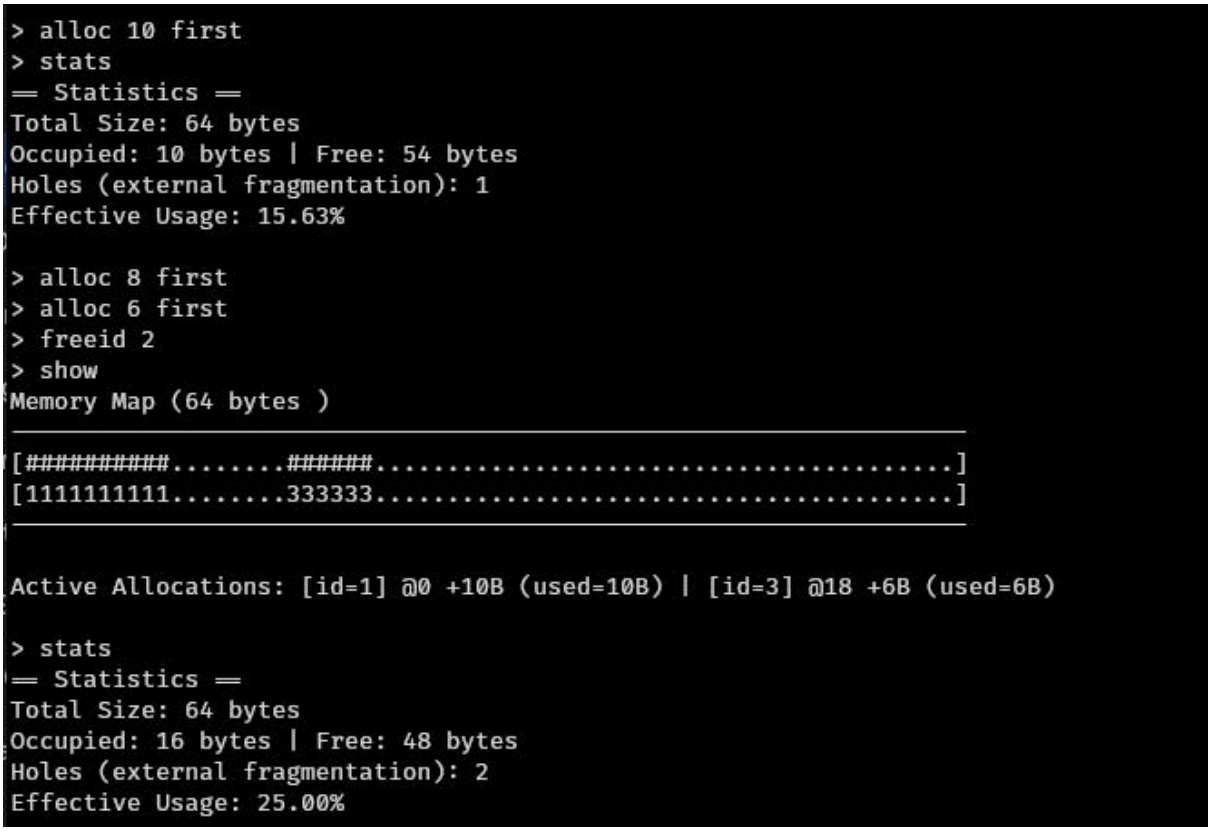
- **Fragmentação Externa**: Contagem de buracos isolados que, embora somem espaço, não são contíguos o suficiente para novas alocações.
- **Uso Efetivo**: Percentual da memória real ocupada por IDs de processos em relação ao tamanho total inicializado pelo comando init.

5. Resultados e Validação Experimental

5.1 Cenário de Teste

Para validar o simulador, foi utilizada uma sequência de operações de alocação e liberação:

1. init 64: Inicialização de um espaço contíguo de 64 bytes.
2. alloc 10 first: Alocação do primeiro bloco (ID 1) no início da memória.
3. alloc 8 first: Alocação do segundo bloco (ID 2) imediatamente após o primeiro.
4. freeid 2: Liberação do segundo bloco, criando um "buraco" de 8 bytes entre o ID 1 e o restante da memória livre.
5. alloc 6 best: Solicitação de 6 bytes utilizando a política de "Melhor Encaixe".



5.2 Comparativo de Comportamento

Algoritmo	Lógica de Seleção	Impacto na Memória
First Fit	Busca linear, primeira lacuna.	Rápido, mas gera acúmulo de uso no início.
Best Fit	Busca exaustiva pela menor lacuna.	Ótimo para espaços pequenos, gera micro-fragmentos.
Worst Fit	Busca exaustiva pela maior lacuna.	Mantém buracos grandes, evita micro-fragmentos.

6. Conclusão

Comportamento do Best Fit: O teste comprovou que o Best Fit prioriza a conservação de grandes blocos livres. Ao escolher o buraco de 8 bytes para uma requisição de 6, ele "poupou" o bloco maior de 46 bytes no final da memória para futuras requisições maiores.

Fragmentação Gerada: Observou-se o surgimento de **fragmentação externa** (o pequeno buraco de 2 bytes). Embora o Best Fit minimize o desperdício imediato, ele tende a deixar esses pequenos fragmentos espalhados, o que justifica a necessidade de uma rotina de **Coalescência** (implementada no seu freeid) para tentar reagrupar esses espaços quando blocos adjacentes são liberados.

O simulador de alocação de memória demonstrou como diferentes políticas de escalonamento de espaço impactam a eficiência do sistema. Enquanto o **First Fit** oferece agilidade, o **Best Fit** e o **Worst Fit** proporcionam um controle mais granular sobre a fragmentação.
