Análise Comparativa de Performance de Cópia de Arquivos com e sem utilização de Buffer.

Israel Efraim de Oliveira[1], José Carlos Zancanaro[2]

[1] Curso de Bacharelado em Ciência da Computação - Universidade do Vale do Itajaí (UNIVALI) – Itajaí – SC – Brasil

{israel.oliveira,jose.zancanaro}@edu.univali.br

Abstract. *In this paper, a comparative analysis of copying methods with and without the buffers was performed. The analysis consists of examining runtime for the various file sizes, called "samples".*

Resumo. *Neste artigo, foi realizada uma análise comparativa de métodos de cópia de arquivos com e sem utilização de buffer. A análise caracteriza-se pela examinação dos tempos de execução para diversos tamanhos de arquivos, denominados "amostras".*

1. Introdução

Neste artigo, é realizada uma análise e comparação de duas implementações de métodos de cópia de arquivos (utilizando streams de bytes) sugeridas na disciplina de Programação no curso de bacharelado em Ciência da Computação. As implementações consistem na exploração da performance obtida ao copiar arquivos com e sem a utilização de um buffer no fluxo de dados.

O objetivo central é demonstrar a diferença geral no tempo de execução com diversos tamanhos de arquivos, evidenciando esta, principalmente, em termos de aceleração total.

Para o desenvolvimento deste artigo, focou-se a definição de uma hierarquia de classes (em Java) que permitisse o uso de polimorfismo para possibilitar a realização do experimento proposto com os diferentes métodos. Além disso, verificou-se a necessidade de implementar recursos internos para armazenamento dos resultados dos testes e parametrização da execução.

2. Definições

2.1. Bytestreams

Os *bytestreams* (fluxo de bytes, em português) são estruturas utilizadas para realizar comunicação em nível de byte.

Os fluxos de bytes são divididos basicamente em fluxos de entrada e saída, que funcionam como conexões que transmitem e recebem dados.

No Java, utiliza-se a classe abstrata **InputStream** para armazenar as diversas implementações de fluxos de entrada, enquanto que a classe abstrata **OutputStream** serve como base para os fluxos de saída. Para trabalhar com arquivos, utiliza-se *FileInputStream* e *FileOutputStream* realizar leitura e escrita, respectivamente. Estas classes quando utilizadas isoladamentes não possuem buffers e portanto, os métodos *read()* e *write()* funcionam byte a byte.

Adiante, existe ainda a possibilidade de implementar buffers que aceleram a entrada e saída de dados através de seu armazenamento temporário. A leitura e escrita

passam a ser realizadas em conjuntos de bytes (8192 por padrão) que, reduzem a quantidade de chamadas ao sistema operacional necessárias.

Instancia-se buffers no Java com as classes *BufferedInputStream* para entrada e, *BufferedOutputStream* para saída de dados.

2.2. Diagrama de Classes

A implementação do código necessário para resolução do problema foi dividida no desenvolvimento dos Copiadores (sem e com utilização de buffer) e da classe responsável por realizar os testes. Ademais, foi necessário também criar um gerador de amostras de diversos tamanhos.

2.2.1. Copiador

Para que fosse possível utilizar do polimorfismo, foi definido uma classe abstrata responsável por proporcionar a implementação base de cópia (apenas transfere os bytes, sem instanciar ou fechar os fluxos).

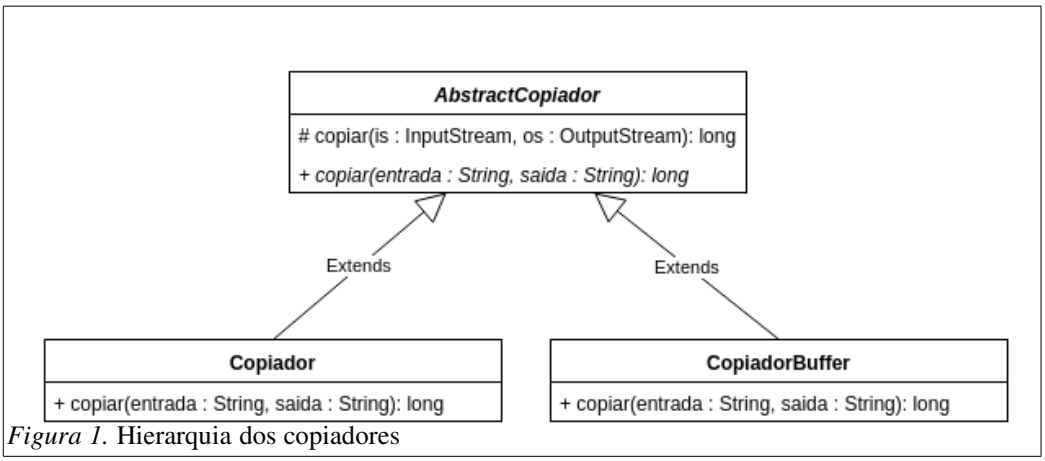

Figura 1. Hierarquia dos copiadores

O *Copiador* implementa a classe abstrata instanciando os fluxos de bytes sem uso de buffer e chamando o método protegido para realizar a transmissão dos bytes, enquanto que o *CopiadorBuffer* instancia fluxos de bytes com utilização dos buffers e se beneficia do método protegido para realizar a cópia dos bytes.

2.2.2. Testador

Responsável por realizar os experimentos, a classe denominada *Testador* recebe uma lista de amostras (estruturas que contém um caminho de arquivo e o tamanho deste) e testa-a com alguma implementação da classe abstrata *AbstractCopiador*.

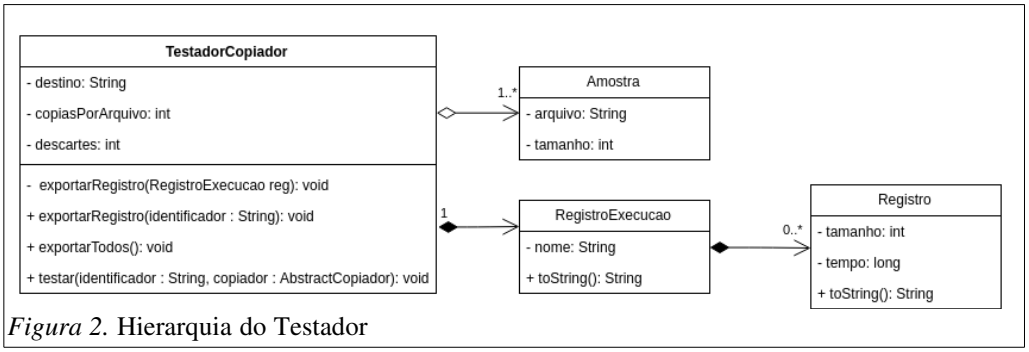

Figura 2. Hierarquia do Testador

O Testador realiza diversas cópias de cada amostra passada e realiza a média de acordo com a quantidade desejada.

Cada vez que o método *testar(…)* for chamado, ele cria um novo registro de execução que pode ser exportado para um arquivo *csv*.

2.3. Ambiente de testes

Para determinar e evidenciar a diferença em performance dos dois métodos propostos, utilizou-se o mesmo ambiente de experimento, sendo este:

- Processador: Intel Core i7 7700HQ (8 núcleos, frequência base ~2.8Ghz)
- RAM: 16 Gbs DDR4
- Armazenamento: 960 Gbs SSD
- OS: Arch Linux (rolling release, kernel 4.18-3), KDE Plasma

Ressalta-se que o espaço de amostragem estava dedicado para obter-se resultados precisos, isto é, o único processo com afinidade no determinado núcleo de execução era o próprio experimento.

3. Comparativo

Nesta seção, apresentamos testes comparativos de análise de tempo de execução (performance) entre ambos os métodos de cópia de arquivo através do uso de fluxo de bytes. O experimento concentra-se na média aritmética do desempenho de tempo de cópia de um conjunto de arquivos que apresentam tamanhos diversificados (em mbs) executado 8 vezes, evidenciando que, por escolha dos autores, os três primeiros testes de cada arquivo são descartados, ignorando, portanto, a possível divergência de tempo devido ao arquivo já estar armazenado na memória principal nas execuções posteriores.

Considerando o supracitado, confere-se, então, nas subseções seguintes os experimentos realizados.

3.1. Copiador sem Buffer

Apesar dos testes serem realizados em um ambiente que dispõe do auxílio de um SSD, o experimento levou um tempo considerável para ser concluído, levando em média, 2 segundos para cada megabyte lido.

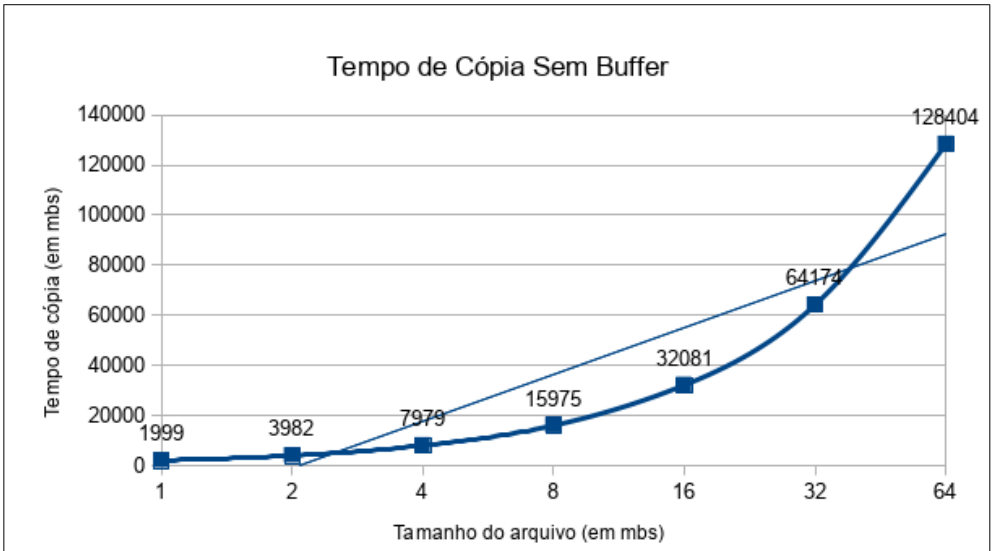

Gráfico 1. Relação de tempo de execução sem uso de buffer.

Os resultados apresentados no Gráfico 1 eram, todavia, esperados, pois a leitura e escrita sem buffer ocorre byte a byte e, portanto, precisam realizar diversas chamadas ao sistema operacional para concluir sua tarefa, gerando, consequentemente, mais lentidão no processo. Considerando que cada byte lido gere uma chamada ao SO, para os 127 megabytes lidos em todo o processo, são necessárias 133.169.152 chamadas de leitura e, a mesma quantidade para escrita, totalizando 266.338.304 chamadas no processo.

Verifica-se, portanto, que a fórmula geral para se obter a quantidade de chamadas para cada megabyte lido/escrito é expressa por:

$$chamadas(x) = 1024 * 1024 * x$$

O crescimento de tempo de execução observado é linear, como demonstrado também no Gráfico 1 através da linha de tendência, que acompanha os dados no eixo Y. Apesar da linha de dados ser análoga a um crescimento exponencial, deve-se lembrar que o tamanho dos arquivos é duplicado a cada valor no eixo X e, portanto, o tempo de leitura deve acompanhar em quantidade.

3.2. Copiador com Buffer

Os experimentos realizados com a utilização de buffer nos fluxos de bytes foram relativamente satisfatórios, se compararmos com os testes anteriores (sem buffer), devido à notável superioridade de desempenho. A média geral de tempo por milisegundo gasto aproxima-se de 6 unidades, isto é, 0,006 segundos.

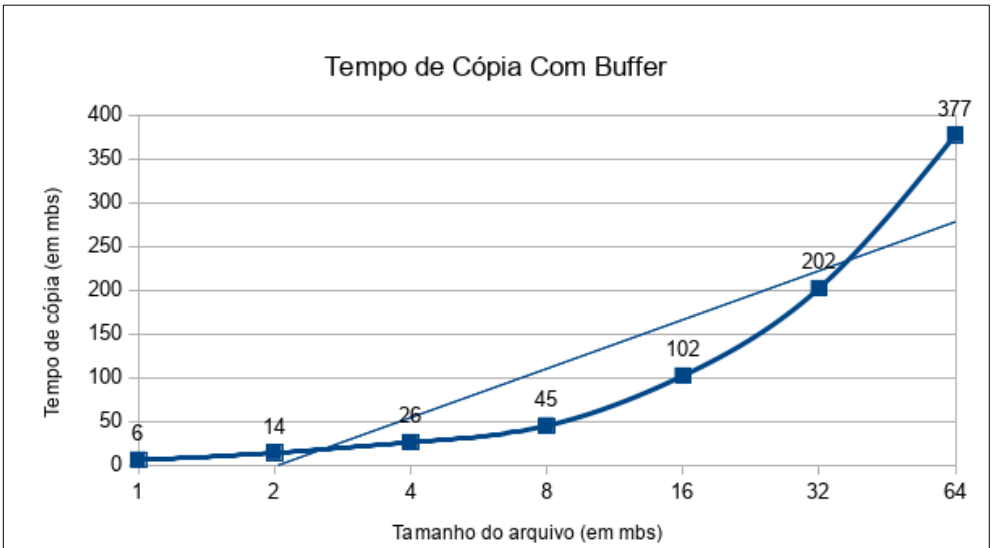

Gráfico 2. Relação de tempo de execução com uso de buffer.

A performance obtida com a utilização de buffer apresentada no Gráfico 2 é notoriamente superior aos testes sem uso de buffer. Isto acontece porque a leitura/escrita não ocorrem byte a byte e sim em conjuntos de bytes, sendo o valor padrão do buffer 8192 bytes.

Para obter o número de chamadas realizadas ao sistema operacional, deve-se utilizar a seguinte fórmula:

$$chamadas(x) = 16256 * x$$

Sendo 16.256 obtido através da razão entre a quantidade de bytes existentes em um megabyte (1048576) e o tamanho do buffer em bytes.

Utilizando a fórmula supratranscrita, obtém-se, portanto, um total de 2.064.512 chamadas ao SO, caracterizando-se aproximadamente 129 vezes mais rápido que o teste sem buffer em questões de chamadas externas.

3.3. Razão

Subsequentemente, foi possível extrair a razão de aceleração entre os dois métodos de cópia de arquivos, utilizando a seguinte fórmula:

$$Aceleração = \frac{sem\ buffer}{com\ buffer}$$

A fórmula referenciada propicia a quantidade de vezes que o método com buffer é melhor que o método sem buffer.

Configura-se no Gráfico 3, portanto, a razão de aceleração observada nos diferentes tamanhos de arquivo.

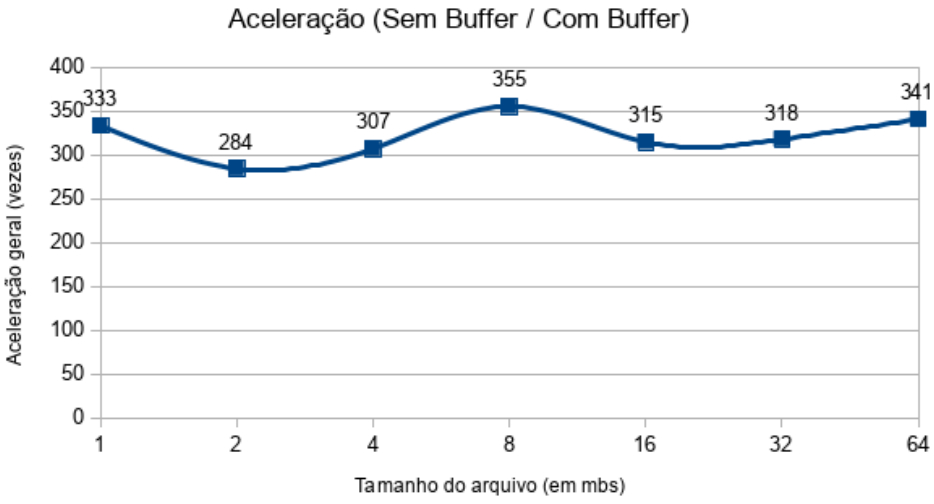

Gráfico 3. Relação de aceleração

Observa-se que o tempo de leitura e escrita obtidos foram muitas vezes melhor nos experimentos realizados com utilização de buffer, isto deve-se ao fato de que para cada megabyte, o copiador sem buffer realizava 1.048.576 chamadas ao SO, enquanto que o copiador com buffer realizava apenas 128.

Realizando a média aritmética de aceleração obtida em todos os tamanhos de arquivos, evidencia-se que a cópia com buffer é aproximadamente 322 vezes mais rápida que o cópia sem utilização de buffer.

4. Conclusão

Com base nos experimentos confeccionados, aferi-se que a utilização de buffer é muito vantajosa em todos os casos, pois proporciona maior velocidade de leitura e escrita dos arquivos, até nos casos em que o arquivo é relativamente pequeno.

Era de se esperar tal diferença, levando em conta que as chamadas ao sistema operacional são muito custosas, pois dependem muito do estado atual do computador em questão de recursos, isto é, acesso à memória principal e armazenamento, e também do quão ocupado o sistema operacional está, pois pode demorar para atender ao chamado. Portanto, menor a quantidade de chamados, maior a velocidade obtida no final do processo.

Em geral, evidencia-se que a utilização de buffer é sempre melhor, porque realiza 8192 menos chamadas por megabyte lido/escrito e beneficia consequentemente, a rapidez com que o processo é executado.

5. Referências

COFFREY, Neil. (2012). InputStream Buffering. Encontrado em <https://www.javamex.com/tutorials/io/input_stream_buffering.shtml>.

BUFFERED Streams. Java Docs. S.l. Encontrado em <https://docs.oracle.com/javase/tutorial/essential/io/buffers.html>.