EP3

MAC0422 - Sistemas Operacionais
Segundo Semestre (2020)

Nome: Pedro Fernandes nUSP: 5948611

Nome: Thiago Jose Benitez Pena nUSP: 6847829

Objetivo

Simular o funcionamento de uma unidade de disco quanto ao que diz respeito a sistemas de arquivos. Desenvolvendo um sistema ágil e funcional que permita diversos comandos de usuários via prompt de comando.

Divisão dos códigos-fontes

- **ep3.c**
  - Código main, lê entrada, inicializa classe driver (interface com o armazenamento: “disco”).
- **driver.cpp** - driver.hpp
  - Declaração da classe driver junto com todo seu funcionamento via funções principais (públicas) e auxiliares (privadas).
- **tools.cpp** - tools.hpp
  - Código com funções auxiliares com propósitos gerais a serem usadas pelo resto do programa.

Funcionamento do programa

Há a necessidade de inicialização com o primeiro comando de interação: o “mount”. Sendo duas as opções:

1. Criar um sistema de arquivos novo, ou seja, com apenas as informações funcionais e pasta root.
2. Ler um sistema já estabelecido e montar o programa sobre ele, isto é, carregar na RAM, informações de FAT e espaço livre.

Após essa etapa inicial, o usuário é livre para utilizar os comandos de interação presentes no enunciado.

Funcionamento do programa - Parte 2

- Optamos por somente utilizar informações básicas de funcionamento (vetor de FAT, free space, nome de sistema, etc.) carregadas em nosso programa.
- Todos os comandos, utilizam o próprio sistema de arquivos para leitura e escrita, logo não houve necessidade de criação de structs e outros sistemas de dados auxiliares para o funcionamento.
- Sendo somente “carregados” pelo programa os blocos (4kb) de certos arquivos pedidos sendo eles únicos ou concatenados caso o arquivos ou diretório exceda o tamanho de um bloco.
- OBS: Para todas as operações, é necessário referenciar o diretório raiz "/" como root/ (ver arquivo LEIAME).

Funcionamento do programa - Parte 3

- Toda procura é feita seguindo caminhos de diretório, sendo algumas decisões de projeto as seguintes:
  - Nome do diretório: root/
  - Todo diretório: termina seu nome com “/”. exemplo: dir1/
  - Todo arquivo: qualquer nome sem “/”. exemplo: arq1
  - Os metadados se encontram no bloco inicial de um diretório.
  - Sobre os metadados dos arquivos* os diretórios possuem o nome e FAT inicial . Tempo de criação, atualização e acesso ficam inseridos no bloco inicial do arquivo.

*Quando percebemos que o correto no FAT seria todos os metadados do arquivo contidos dentro da pasta pai, ao invés de nome e FAT somente, já tínhamos escrito 90% do código. Então optamos por deixá-lo como está. Contudo, pesquisamos sobre o assunto e funcionamento do FAT nesse sentido para não repetirmos o erro.

Funcionamento do programa - Parte 4

- Outras decisões de projeto:
  - Representamos o espaço de um byte livre como @. Dessa forma é possível encontrar facilmente um bloco pela posição de seu caractere inicial.
  - A separação de informações entre os elementos contidos em um bloco (metadados e conteúdo) é feita por “|”.
  - As duas primeiras linhas contém os blocos do gerenciamento de espaço livre e da fat.
  - A separação dos demais blocos é feita por “\n” (implícito na quebra de linha do arquivo texto).
  - O espaço livre é representado por 7 blocos, constituindo 28000 bytes, mas somente 24956 utilizados. (não há separação desses 7 blocos)
  - Representação de bloco utilizado = 1, bloco livre = 0.

Funcionamento do programa - Parte 5

- Outras decisões de projeto:
  - Tabela FAT é representado por 38 blocos, constituindo 152.000 bytes, mas somente 149.729 utilizados.
  - Números FAT são representados por 5 bytes para padronização

    exemplo. |00005|00001|01206|0...

  - representação do espaço final da FAT: |-0001|
  - representação do espaço vazio da FAT: |-0002|

Método de experimento

- Optamos, pelas mesmas máquinas usadas nos EPs anteriores, todavia, agora, com uma diferença. Para ilustrar as particularidades de acesso ao disco pelo S.O. fizemos os testes da máquina “B” simulando um uso não previsto do sistema de arquivos, explicando melhor, ficamos constantemente abrindo o “sistema de arquivos virtual” (arquivo texto montado) durante seus processos de escrita e remoção. Isso resultou em ocasionais resultados de experimento mais demorados (outliers).

- Todos os arquivos a serem copiados foram montados previamente sendo as repetições do experimento sempre feitas sobre arquivos idênticos.

Máquinas do experimento

Máquina [A]

- 1 CPU - 12 processadores lógicos

  Ryzen 1600 - 3.5Ghz (Overclock)
  - 6 núcleos físicos - 12 lógicos

- 16 GB RAM 2900mhz
- SSD
- Ubuntu 16.04.7 LTS

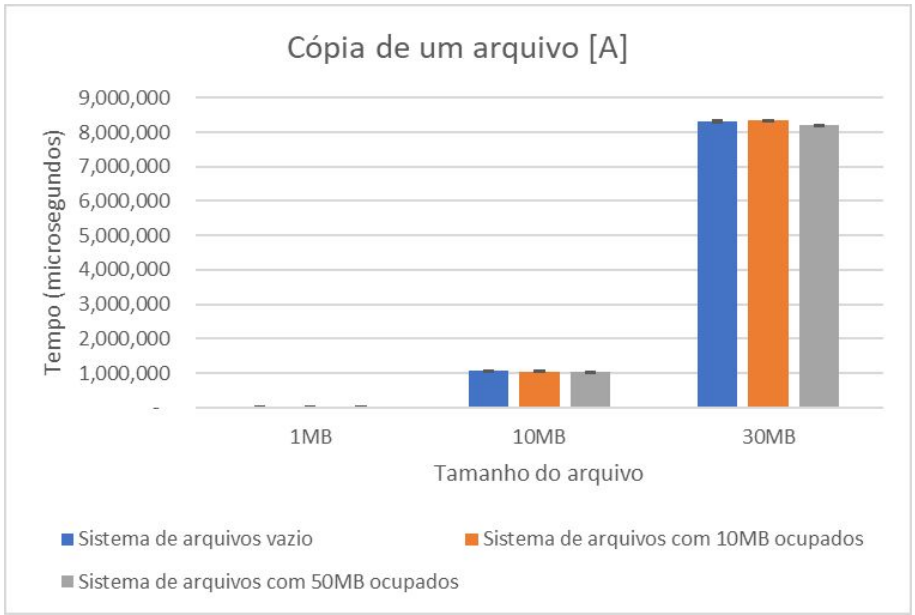
Máquina [B]

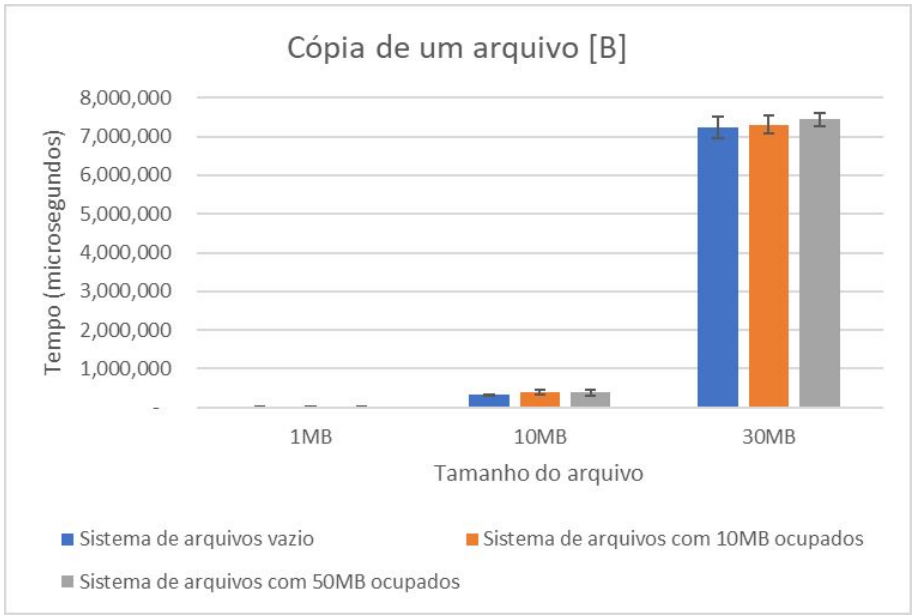
- 1 CPU - 8 processadores lógicos

  (Ryzen 3300X - 3.9Ghz
  4 núcleos físicos - 8 lógicos)

- 16 GB RAM 3000mhz
- SSD
- Ubuntu 20.04.1 LTS

Gráficos - Cópia de um arquivo (experimentos 1, 2 e 3)



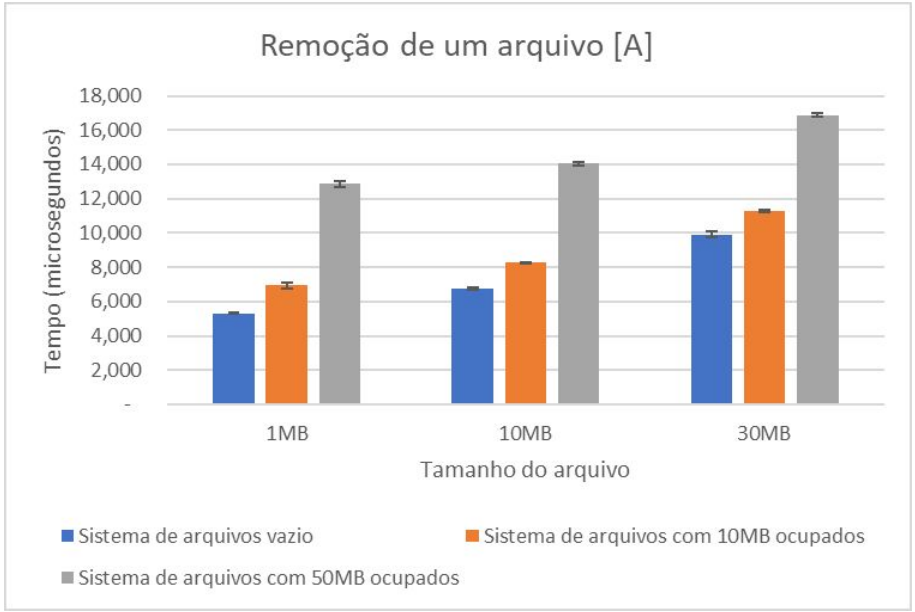


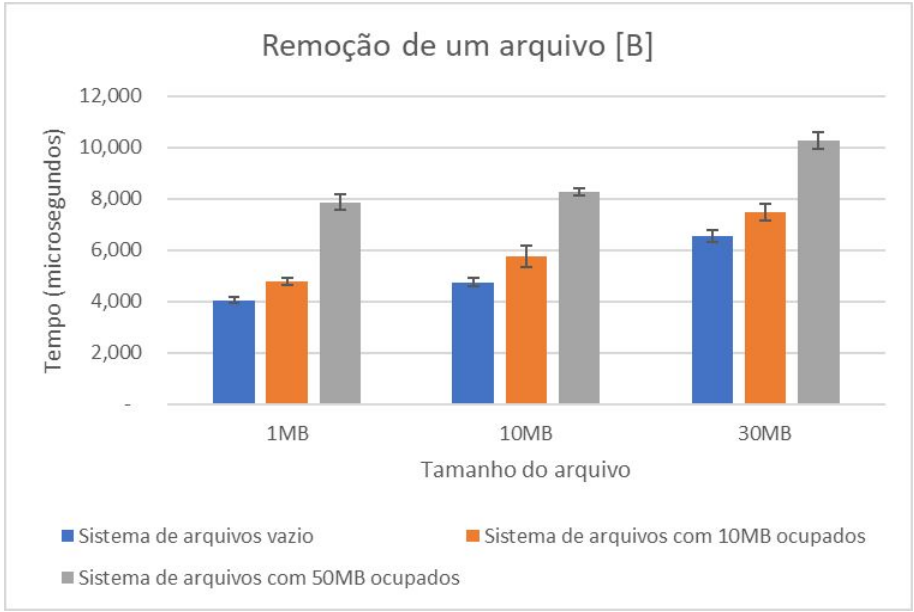
Conclusões - Cópia de um arquivo (experimentos 1, 2 e 3)

Percebemos que o fator mais expressivo das diferenças foram o tamanhos dos arquivos, ou seja, a cópia dos arquivos foi um processo lento.

O fato dos sistemas estarem já ocupados por diferentes valores gerou também diferenças no desempenho da cópia, contudo, nem de longe tão determinantes quanto o tamanho do arquivo ocupado.

Gráficos - Remoção de um arquivo (experimentos 4, 5 e 6)



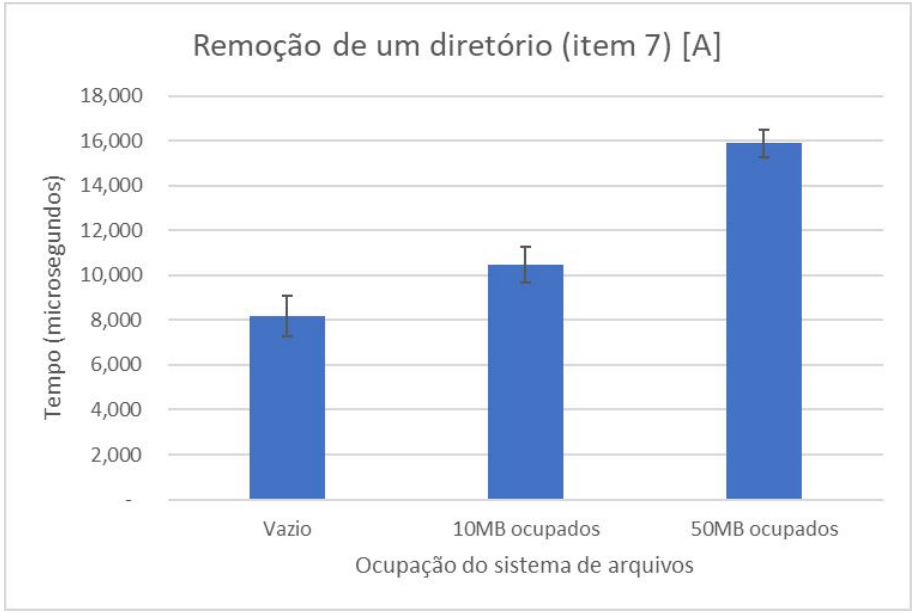


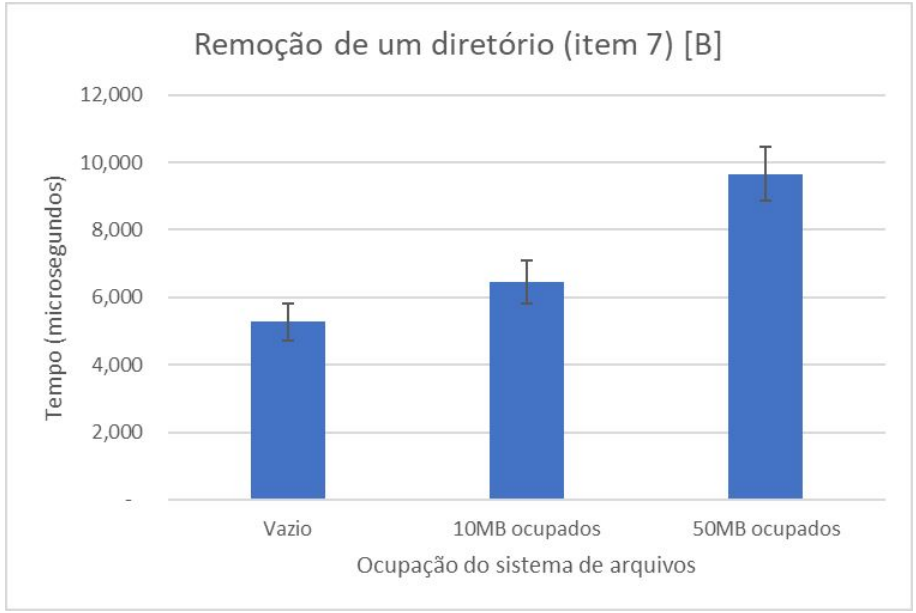
Conclusões - Remoção de um arquivo (experimentos 4, 5 e 6)

Os resultados das remoções foram parecidos quando o da cópia no sentido de que o fator primordial no consumo de tempo é o tamanho do arquivo a ser removido. Todavia, aqui vimos um maior expressividade, nas diferenças de tempo, de acordo com o quanto o sistema de arquivos já está ocupado.

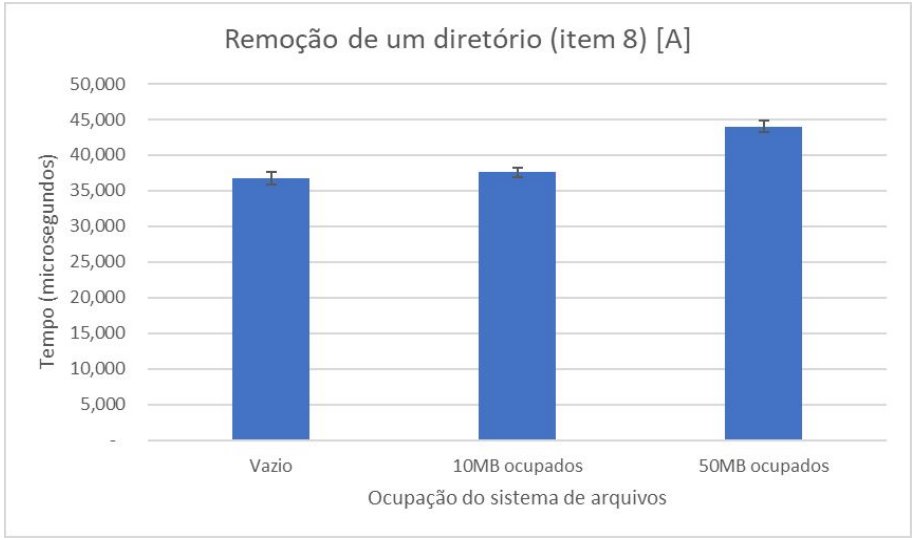
Acreditamos que essa diferença, em relação a cópia, seja devido ao caráter recursivo da função de remoção, isto é, várias chamadas levaram a percorrerem longos caminhos várias vezes em uma situação de disco mais cheio.

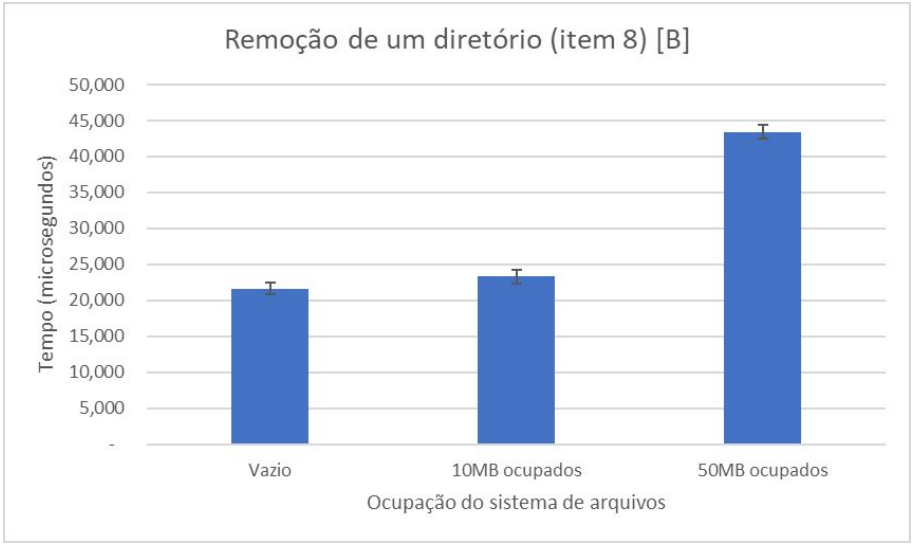
Gráficos - Remoção de um diretório (experimento 7)





Gráficos - Remoção de um diretório (experimento 8)





Conclusões - Remoção de um diretório (experimento 7 e 8)

Os resultados aqui estiveram dentro do previsto, o experimento 8 demandou bem mais tempo de uso devido as centenas de arquivos em seus diretórios*.

*No experimento 8 criamos 100 arquivos em cada um dos 30 diretórios da hierarquia, totalizando 3000 arquivos que ocupam apenas 1 bloco.

Conclusões - Máquina A e B

Vimos, como era esperado, que o fato da máquina A estar sendo usada durante os experimentos, mesmo ambas tendo diversos núcleos, ocasionou em uma amplitude de erro muito maior. Percebemos com isso que, o disco, devido ao caráter de sua interação com o sistema ser unitária, qualquer acesso ao arquivo sendo utilizado, mesmo que seja somente uma simples leitura, já interfere na fila de processamento.

Conclusões Finais

Esse exercício programa nos ajudou mentalmente a fixar um paradigma de funcionamento de sistemas de arquivos, tão bem quanto às consequências e ramificações que cada decisão de implementação afeta tanto a sua organização quanto na velocidade de processamento.

Referências

Várias referências de uso das biblioteca fstream, fstream, assim como diferentes técnicas para manejo de strings
inspiradas nos sites:

- https://pt.stackoverflow.com/
- https://man7.org/
- https://linux.die.net/