Escola de Artes, Ciências e Humanidades

ACH2044

Sistemas Operacionais

Relatório EP 2

Autores:
Bernardo Chagas Araújo — 10367210
Daniel Bolsonaro Vaz Filho — 10346867
Leonardo Soares Santos — 10284782

23 de Novembro de 2018

Conteúdo

Lista de Figuras

1 Código Fonte

1.1 Pacote preferencia_Leitor

Nesse pacote foram usados algoritmos com base no uso de semáforos para evitar a utilização de leitores ou escritores em horas indevidas. Ele favorece os leitores de modo que, sempre que um leitor já estiver lendo, é possível outro leitor fazer a leitura, mas caso algum escritor venha a requisitar o uso do banco de dados, ele fica em estado de espera até que nenhum leitor ou escritor esteja fazendo uso do banco.

1.2 Pacote preferencia_Escritor

O pacote possui um algoritmo derivado da implementação do pacote preferencia_Leitor, com algumas alterações na classe Threads. No caso, foram adicionados 1 semáforo para controle de fluxo, além disso, alguns semáforos passaram a ter noção de justiça (new Semaphore(int permits, boolean fairness) que faz com que, as threads que solicitaram os créditos antes, serão atendidas primeiro, criando uma situação de FIFO.

1.3 Pacote regiaoCritica

O Pacote regiaoCritica foi o primeiro implementado por ser o pacote mais simples, uma vez que não tem preferencia sobre algum tipo de thread. Ele apenas espera todas as threads terminarem para avançar no programa usando a função join() da classe Thread do java.

1.4 Classes

Cada apacote possui essas três classes com algumas pequenas variações, mas elas basicamente possuem as mesmas funções e executam quase as mesmas operações.

1.4.1 Main

Tem como principal função a criação dos arquivos de log e chamadas da classe CriaThreads com as proporções de cada leitor e escritor e deixando o banco de dados como uma variável global.

1.4.2 CriaThreads

Essa classe cria o arranjo de threads e efetivamente roda todas as threads em uma determinada proporção de Leitores e Escritores, para melhor visualização do gráfico as funções que criam e rodam a thread são rodadas 100 vezes, além disso, é nessa classe que se determina quanto tempo demora (em uma média de 100 execuções) para rodar esse algoritmo com a proporção passada pela classe Main. Nessa classe estão localizados os semáforos para as implementações com semáforo

1.4.3 Threads

A classe das Threads estende à classe Thread do java e tem como função efetivar a leitura e escritura dos dados em si. Nela se encontra o algoritmo responsável pela resolução do problema dos Escritores e Leitores com preferência de leitores.

1.5 Classe Testes

A classe Testes.java, como o próprio nome já diz, foi feita única e exclusivamente para a bateria de testes, ela não tem nenhum pacote e importa todos os três, e faz chamadas aos métodos main de cada classe.

2 Resultados obtidos e Algoritmos usados

Todos os resultados foram obtidos com o tempo de execução das threads em milisegundos em função do número de leitores. O teste realizado executou 100 vezes cada sistema realizando a média aritmética para a boa visualização dos gráficos, sendo assim, esses gráficos são resultado de 30000 threads rodando em 3 sistemas diferentes durante 1 hora (não ao mesmo tempo).

O computador usado possui processamento lógico e aritmético superior, além de contar com bom armazenamento temporário, por isso os resultados ficaram baixos e com pouca variação.

2.1 Implementação Região Crítica

Nessa implementação, o resultado dos tempos foi continuo. Com uma variação pequena de 2 ou três milisegundos.

Para essa implementação foi usado o método join() da classe Thread para sinalizar a thread seguinte que deveria esperar a anterior terminar antes de começar, simulando assim a função da região crítica.

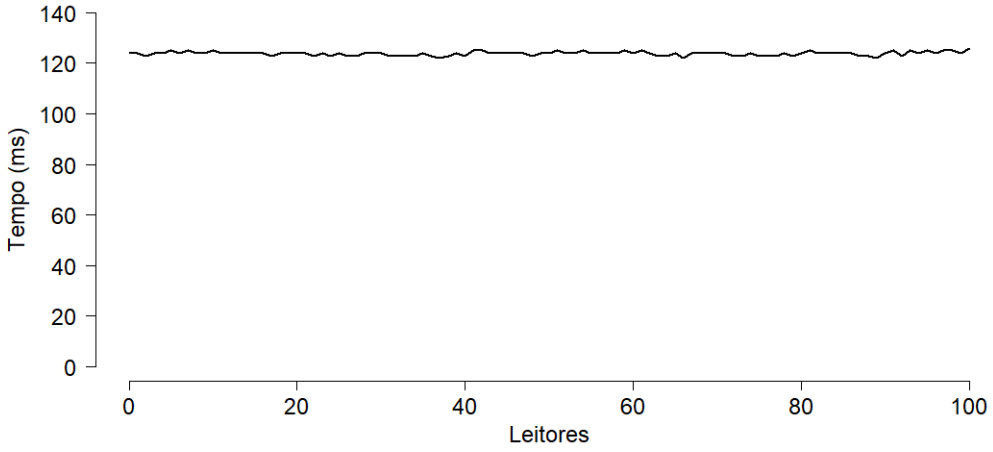


Figura 1: Gráfico da implementação da Região Crítica

Como podemos ver no gráfico 1, a operação de join() faz com que todas as threads demorem o mesmo tempo, com variação de 3 milisegundos de diferença em tempo de execução, confirmando o esperado.

2.2 Implementação da preferência do leitor

Essa implementação proporcionou um gráfico linear decrescente com tempo máximo ma região de 100 escritores.

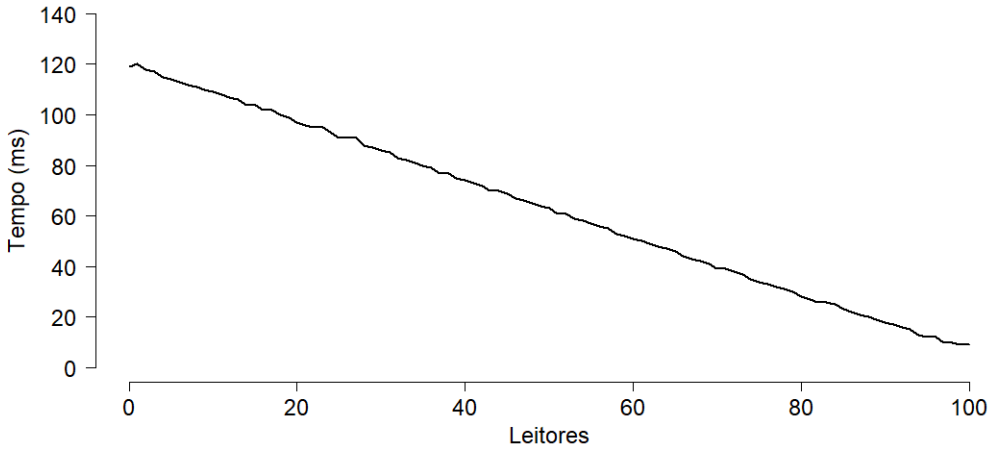


Figura 2: Gráfico da preferência do leitor

O algoritmo usado para criação das estruturas de threads por semáforos foi sugerido por TANENBAUM, A.S. em seu livro Sistemas Operacionais Modernos, 2a ed - A.S. Tanenbaum (2003).

Esse resultado do gráfico 2 foi satisfatório para o esperado do algoritmo, que propõe que escritores não podem usar o banco de dados se algum leitor ou escritor estiver fazendo o uso dele, entretanto, os leitores podem fazer uso enquanto algum leitor já estiver usando o banco de dados, mas não quando um escritor o estiver fazendo. Esse algoritmo propõe a preferência para os leitores, por isso o nome da classe

2.3 Implementação da preferência do escritor

Já no terceiro sistema implementado, existe uma parábola significativa no gráfico, que significa a preferência por parte dos escritores.

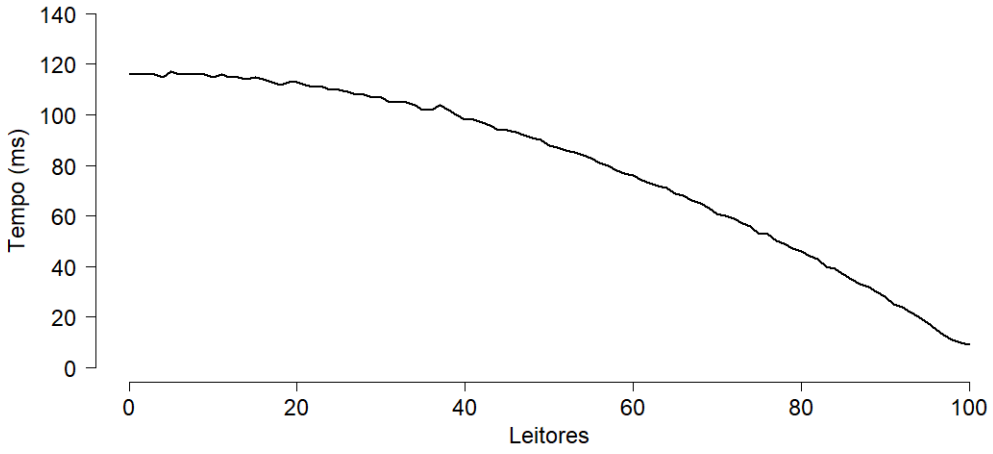
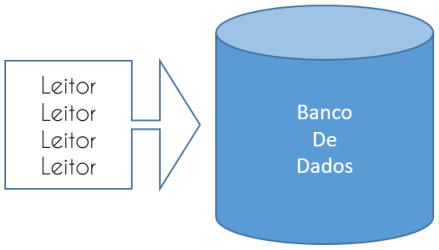


Figura 3: Gráfico da preferência do escritor

Essa parábola se deve a implementação voltada a preferenciar os escritores que, além de não sofrerem starvation, ainda tem preferência de uso do banco dados. Para facilitar o entendimento, será esclarecido o funcionamento do algoritmo.

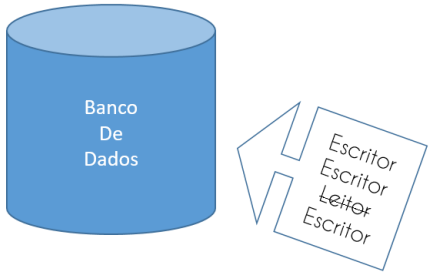
2.3.1 Leitor e leitor

Quando existem 2 leitores, o primeiro na "fila"para o uso do banco de dados assinala um semáforo dizendo que já existe pelo menos um leitor usando o banco e isso libera o leitor seguinte para usar o banco ao mesmo tempo.



2.3.2 Escritor e escritor

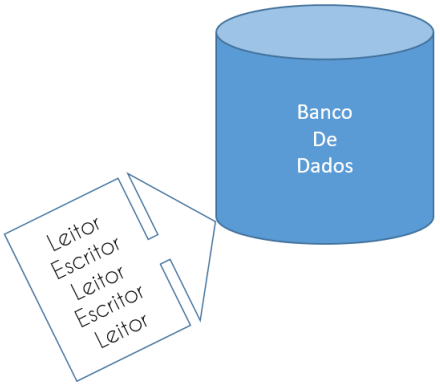
Se existe um escritor já fazendo acesso ao banco de dados, um escritor que venha a pedir acesso tem preferência ao solicitar o semáforo que bloqueia o Leitor, por uma condição de Fairness, que cria uma fila FIFO para as threads (Saiba mais em: `https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html`).



Tendo isso em mente, é destacado que, ao entrar um escritor na fila, ele é "passado na frente"de todos os leitores por uma condição de preferencia.

2.3.3 Leitores e escritores

Suponha que existam 5 threads, nas quais 3 são threads leitoras e 2 são thread escritoras. Elas estão organizadas da seguinte forma:



O que aconteceria é que o primeiro leitor iria entrar e começar a ler, nesse momento, o primeiro escritor viria e solicitaria acesso ao banco de dados, impedindo que próximos leitores entrassem para fazer o acesso, no momento em que o leitor

saísse e o escritor começasse a escrever, chegariam um leitor e um escritor, nessa hora, o algoritmo escolhe o escritor, pois já existe um escritor escrevendo no banco de dados, porém, ele não começa imediatamente, ele tem que esperar o primeiro escritor terminar seu trabalho.

Após o trabalho de ambos escritores ser finalizado, o segundo leitor vai verificar que não existe nenhum escritor pendente na fila de threads e vai pegar o banco de dados para si. Ao mesmo tempo que isso acontece, o ultimo leitor chega na fila para solicitar acesso ao banco e, vendo que já existe um leitor ativo, ele entra no banco imediatamente.

Desse jeito, é possível perceber que há um "gap"de agilidade no tempo de execução, pois tem a proibição de uso direto por parte dos leitores e por isso existe essa "barriguinha"no gráfico, principalmente na parte em que leitores e escritores estão com proporção de 50% cada. Isso fica mais visível na próxima sub-seção.

2.4 Diferença das preferências

Ao analisar o gráfico da figura 4 percebe-se, quando a diferença passa do leitor para o escritor, a pequena "barriguinha"que se forma quando à medida que vão existindo, na mesma proporção, leitores e escritores.

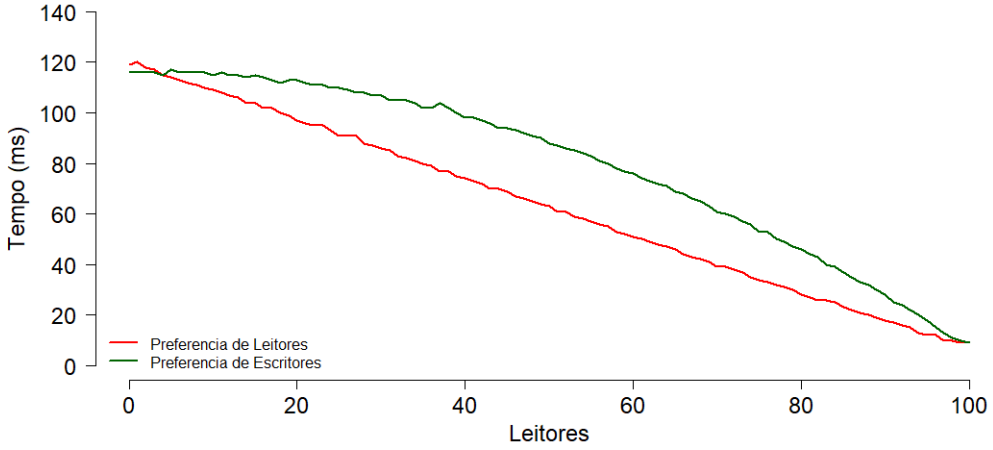


Figura 4: Gráfico das implementações preferências

3 Conclusão sobre os resultados

Com todos os gráficos sobrepostos, é possível perceber a gritante diferença entre os algoritmos, mas não por diferença na eficiência, mas sim, por diferença na função de cada um deles e do problema.

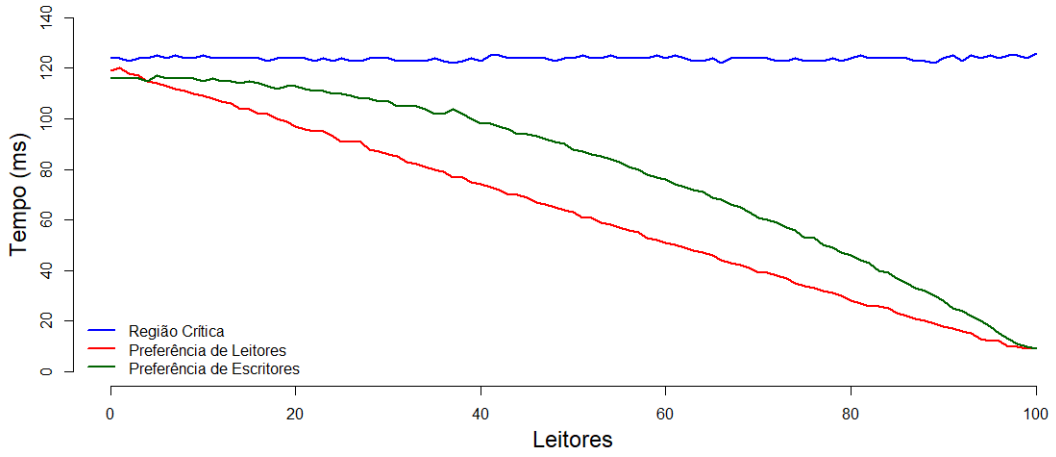


Figura 5: Gráfico de todas implementações

Com a zona de Região Crítica, o problema não importava quem já estava lá, lendo ou escrevendo, e quem estava esperando para entrar, ele simplesmente esperava quem estava fazendo terminar de fazê-lo. Já na hora das preferências, havia concorrência pelo acesso ao banco de dados principalmente pelo escritor, por isso, na medida em que os escritores foram aumentando, o tempo também foi.

4 Observações finais

O trabalho pode ser encontrado no git: `https://github.com/ddeniel1/EP2-SO`. Toda a metodologia usada foi com base na descrição do problema do ep, disponível no git. Os algoritmos prontos que foram implementados diretamente na classe Threads tem autoria de terceiros, todavia, todas as demais classes foram desenvolvidas por nós, de acordo com o problema a ser resolvido.