Sistema Produtor-Consumidor com Buffer Compartilhado - Uso de Múltiplas Condições com um Lock

STATUS **DESENVOLVIDA**

Objetivo do Projeto

Este projeto implementa um sistema Produtor-Consumidor em Java, utilizando ReentrantLock e Condition para garantir a sincronização entre threads. O objetivo é simular a leitura de um arquivo de texto por um produtor, que insere as linhas em um buffer compartilhado, enquanto consumidores processam essas linhas.

Estrutura do Código

O código é composto pelas seguintes classes:

Buffer: Representa o buffer compartilhado entre o produtor e os consumidores. Utiliza ReentrantLock para controle de acesso e Condition para sincronização. Producer: Simula a leitura de um arquivo e insere as linhas no buffer. Consumer: Consome as linhas do buffer e realiza um processamento simulado. FileMock: Gera um arquivo simulado com linhas de texto aleatórias. Main: Inicializa o sistema, criando um produtor e múltiplos consumidores.

Funcionamento

O produtor lê as linhas do FileMock e as insere no buffer até que ele esteja cheio. Os consumidores retiram as linhas do buffer e as processam. Se o buffer estiver cheio, o produtor aguarda até que haja espaço disponível. Se o buffer estiver vazio, os consumidores aguardam até que novas linhas sejam inseridas. O sistema continua até que todas as linhas tenham sido processadas.

🗒️ Cenários teste

1. Produtor único e um consumidor

Verifica se as linhas são lidas e processadas corretamente sem concorrência significativa.

=======

2. Produtor único e múltiplos consumidores

Testa o compartilhamento de carga entre os consumidores. Avalia o comportamento de sincronização e paralelismo.

🔗

3. Aumento do tamanho do buffer e do número de linhas

Mede o impacto no desempenho e verifica se a sincronização continua funcionando corretamente.

🔧 Conclusão:

🔗

Este projeto implementa um modelo eficiente de Produtor-Consumidor, garantindo sincronização adequada com ReentrantLock e Condition. Foram realizadas as seguintes modificações:

Comentários detalhados para melhor compreensão do código. Documentação das saídas esperadas para auxiliar nos testes. Incrementação do controle de estado com hasPendingLines(), garantindo que os consumidores parem corretamente ao término do processamento. Depuração com mensagens de console indicando o fluxo de execução.

Este sistema pode ser expandido para suportar diferentes tamanhos de buffer, número variável de consumidores/produtores e novos cenários de teste para análise de concorrência.

💻 Desenvolvedor

🔗

Marcio Fonseca
👨‍💻

=======