Relatório: Uso de Múltiplas Condições para Coordenação de Produtor e Consumidor

1. Objetivo

Este exemplo demonstra a utilização de múltiplas condições com `ReentrantLock` para coordenar a comunicação entre threads produtoras e consumidoras. O sistema simula um buffer onde o produtor insere linhas lidas de um arquivo simulado (FileMock) e os consumidores removem e processam essas linhas. O uso de condições garante que o produtor aguarde quando o buffer estiver cheio e os consumidores aguardem quando o buffer estiver vazio, mantendo a integridade dos dados.

2. Estrutura do Código

- **Buffer**:
  Gerencia o armazenamento de linhas usando uma lista encadeada. Possui duas condições:
  - `lines`: sinaliza que há linhas disponíveis para leitura.
  - `space`: sinaliza que há espaço disponível para inserção. Além disso, utiliza um booleano `pendingLines` para indicar se ainda há linhas a serem inseridas.
- **Consumidor**:
  Implementa `Runnable` e representa uma thread consumidora que remove linhas do buffer enquanto houver linhas pendentes, processando cada linha com um atraso aleatório.
- **FileMock**:
  Simula um arquivo contendo um número definido de linhas com conteúdo aleatório. Fornece métodos para verificar a disponibilidade e obter as linhas.
- **Produtor**:
  Implementa `Runnable` e representa uma thread produtora que lê linhas do `FileMock` e as insere no `Buffer`. Ao terminar, sinaliza que não há mais linhas pendentes.
- **Principal**:
  Inicializa o sistema criando uma instância de `FileMock`, um `Buffer`, uma thread produtora e três threads consumidoras, permitindo a execução concorrente e sincronizada do produtor e dos consumidores.

3. Fluxo de Execução

1. **Inicialização**:
   - Um `FileMock` é criado para simular um arquivo com 101 linhas.

- Um `Buffer` com capacidade para 20 linhas é instanciado.

2. **Produção**:
   - A thread produtora lê linhas do `FileMock` e insere cada linha no `Buffer`.
   - Se o buffer estiver cheio, o produtor aguarda até que haja espaço.
3. **Consumo**:
   - As threads consumidoras removem linhas do buffer enquanto o método `hasPendingLines()` indicar que ainda há dados a serem processados.
   - Se o buffer estiver vazio, os consumidores aguardam até que novas linhas sejam inseridas.
4. **Sinalização e Sincronização**:
   - As condições `lines` e `space` são utilizadas para coordenar as operações de inserção e remoção, garantindo que o produtor e os consumidores se comuniquem de forma eficiente.

4. Exemplo de Execução

Saída do Console:

```
Mock: 101
Produtor insere linha... (saída variada, informando o número de linhas restantes)
Consumidor 0: Line Readed: 19
Consumidor 1: Line Readed: 18
Consumidor 2: Line Readed: 17
...
```

5. Conclusão

Este exemplo evidencia a eficácia do uso de múltiplas condições com ReentrantLock para gerenciar a comunicação entre threads em um cenário produtor-consumidor. Ao sincronizar o acesso ao buffer com as condições lines e space, o sistema assegura que o produtor aguarde quando o buffer estiver cheio e os consumidores quando estiver vazio, evitando condições de corrida e garantindo a integridade dos dados compartilhados. Essa abordagem é fundamental para aplicações que necessitam de uma comunicação eficiente entre threads concorrentes.