Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Campus Apucarana
Bacharelado em Engenharia de Computação

Nome: Deivid da Silva Galvão, 2408740
Nome: João Vitor Nakahodo Yoshida, 2419904
Professor: Lúcio Rocha Agostinho

Lista de Exercícios 3 - Teoria (Atividade em DUPLA)

1) **Atomicidade** → Cada operação de transação, podendo ser read, write, update ou delete) são tratadas como uma única ação, ou seja se qualquer parte do processo falhar todo processo não é consolidado, essa propriedade é importante, pois garante que todas as operações sejam completadas com sucesso ou nenhuma delas seja aplicada, dessa forma previne a corrupção de dados durante a operação.

Consistência → A consistência assegura que alterações feitas no banco de dados não violem as regras de restrições predefinidas, dessa forma garantindo que o banco de dados nunca viole suas restrições de integridade, mantendo a precisão e a validade dos dados.

Isolamento → O isolamento garante que as operações de uma transação sejam isoladas das operações de outras transações, ou seja isso permite que sejam executadas de forma concorrente sem que haja problemas de leituras sujas e sem interferências.

Durabilidade → A durabilidade assegura que uma vez que foi dado o commit (consolidação da transação) as alterações são permanentes e persistem no banco de dados mesmo que o sistema falhe, isso garante que os dados não sejam perdidos após a confirmação de transação, o que proporciona mais confiabilidade e segurança aos dados armazenados.

2)

2.1) H5 = w1[x] r2[x] w2[x] abort1 abort2

- **Não é recuperável,** pois a transação 2 lê o valor escrito por t1 antes dele abortar.
- **Não evita abort em cascata,** pois quando t1 aborta t2 também tem que ser abortado.
- **Não é estrita,** pois t2 lê e escreve antes de t1 confirmar ou abortar .

2.2) H4 = W1[X] commit1 r2[x] w2[X] commit2

- **É recuperável,** pois t2 lê o valor escrito por t1 após t1 ter confirmado.
- **Evita abort em cascata,** pois t2 lê o valor de x após t1 ter confirmado.
- **É estrita,** pois t2 lê e escreve depois de t1 confirmar.

2.3) H3 = w1[X] w2[x] commit1 r2[x] w2[y] commit2

- **É recuperável,** pois t2 lê o valor escrito por t1 após t1 ter confirmado.
- **Não evita abort em cascata,** pois t2 tem dependência de t1.
- **Não é estrita,** pois t2 escreve antes de t1 confirmar.

2.4) H2 = w1[x] w1[y] r2[y] w2[y] commit1 commit2

- **É recuperável.**
- **Não evita abort em cascata,** pois t2 tem dependência de t1.
- **Não é estrita,** pois t2 lê antes de t1 confirmar.

2.5) H1 = w1[x] w1[y] r2[y] w2[y] commit2 commit1

- **Não é recuperável,** pois t2 está lendo um valor ainda não consolidado.
- **Não evita abort em cascata,** pois t2 tem dependência de t1.
- **Não é estrita,** pois t2 lê antes de t1 confirmar.

3) 3.1) H5 r1(x), w1(x,2), Commit1, r2(x), w2(x,3), Abort2

- **É recuperável,** pois t2 está lendo um valor já consolidado por t1.
- **Evita abort em cascata,** pois t1 consolida antes de t2 acessar.
- **É estrita,** pois t2 lê depois de t1 confirmar.

3.2) H1 = W1(x,2), r2(x), w2(y,3), Abort1

- **É recuperável,** pois as transações podem ser desfeitas com um abort em cascata.
- **Produz abort em cascata,** pois t2 tem dependência de t1.
- **Não é estrita,** pois t2 lê antes de t1 confirmar.

3.3) H3 = w1(x,2), Abort1, w2(x, 3), Commit2

- **É recuperável,** pois t2 está escrevendo depois de t1 ter abortado.
- **Não produz abort em cascata,** pois t1 aborta antes de t2 acessar.
- **É estrita,** pois t2 escreve depois de t1 abortar.

3.4) H2 = w1(x,2), r2(x), w2(y,3), Commit1, Commit2

- **É recuperável.**
- **Produz abort em cascata,** pois t2 depende de t1.
- **Não é estrita,** pois t2 lê antes de t1 confirmar.

3.5) H1 = w1(x,2), r2(x), w2(y,3), Commit2

- **Não é recuperável,** pois t2 está lendo antes de t1 ter confirmado.
- **Produz abort em cascata,** pois t2 depende de t1.
- **Não é estrita,** pois t2 lê antes de t1 confirmar.

4)

a) Leitura suja → Ocorre quando uma transação lê dados que foram modificados e que não foram confirmados por outra transação.

Exemplo: **w1[x] r2[x] abort1**

onde t2 lê o “x” sem ter sido confirmado por t1.

b) Cancelamento em cascata → Ocorre quando uma transação é abortada e por

consequência outras transações que tinham dependência a ela também precisam ser abordadas.

Exemplo: **w1[x] r2[x] w2[x] abort1**

Nesse caso t2 precisa abortar a sua leitura e escrita, pois tinha dependência de t1 e ele foi abortado.

c) Gravação prematura → Ocorre quando uma transação escreve dados e depois cancela deixando-o inconsistente.

Exemplo: **r1[x] w1[x] r2[x] w2[x] abort2**
Nesse exemplo, t1 não confirmou nem abortou as primeiras operações de escrita em "x" e t2 fez uma operação de leitura e escrita em "x", ou seja t2 escreveu sem garantir que as operações anteriores fossem concluídas e deu abort gerando uma inconsistência .

d) Leitura prematura → Ocorre quando uma transação lê dados modificados por outra transação antes de ela ter dado commit.

Exemplo: **w1[x] r2[x] abort1**
Nesse caso, t1 faz a leitura antes de t1 confirmar a modificação feita em x, ou seja fez uma leitura prematura e por consequência não é recuperável, levando a um abort em cascata.

e) Atualização perdida → Ocorre quando 2 transações simultâneas fazem a leitura e posteriormente a escrita resultando na perda de uma das atualizações, visto que uma sobrescreve a outra.

Exemplo: **r1[y] r2[y] w1[y] w2[y]**
Nesse caso o conteúdo escrito por t1 será sobrescrito por t2 e consequentemente essa atualização feita por t1 será perdida.

5)

H = r1[y] w1[y] w1 [x] w2[y] r2[x] w2 [x] w1[u] c1 c2

No exemplo, t1 adquire um bloqueio de leitura em "y" (fase de crescimento), lê "y", adquire um bloqueio de escrita em "y" (fase de crescimento), escreve em "y", adquire um bloqueio de escrita em "x" (fase de crescimento) e realiza a escrita. As operações de t2 (escrita em "y", leitura e escrita em "x") são colocadas em uma fila FIFO para serem executadas posteriormente. Por fim t1 adquire a tranca de escrita

em "u" e escreve em “u”, as modificações de t1 são consolidadas e as trancas de leitura e escrita são liberadas (fase de encolhimento). Então, t2 é liberado para acessar "y", adquire um bloqueio de leitura em "y", lê "y", adquire um bloqueio de escrita em "y", escreve em "y", adquire um bloqueio de escrita em "x", realiza a escrita e libera as trancas (fase de encolhimento) após consolidar as modificações em "y" e "x". Isso garante que duas transações não acessem o mesmo item simultaneamente, evitando conflitos de dados e processando as operações na ordem submetida. Após a liberação das trancas, o processo não pode travar novamente para novas operações.

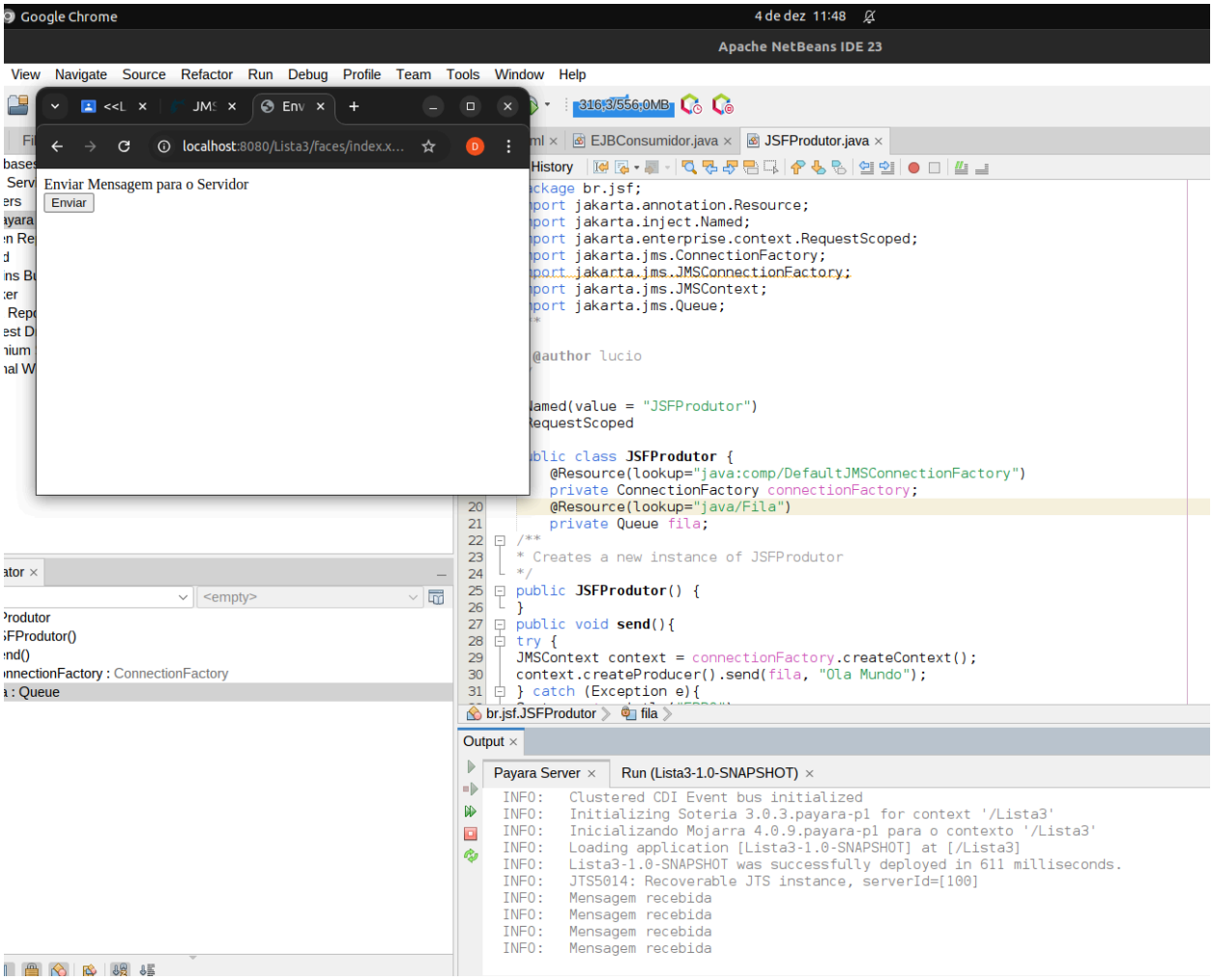
O esquema final usando o algoritmo de 2PL fica assim:
rl1[y] r1[y] wl1[y] w1[y] wl1[x] w1[x] wl1[u] w1[u] commit1 ru1[y] wu1[y] wu1[x] wu1[u] wl2[y] w2[y] rl2[x] r2[x] wl2[x] w2[x] commit2 wu2[y] ru2[y] wu2[x]

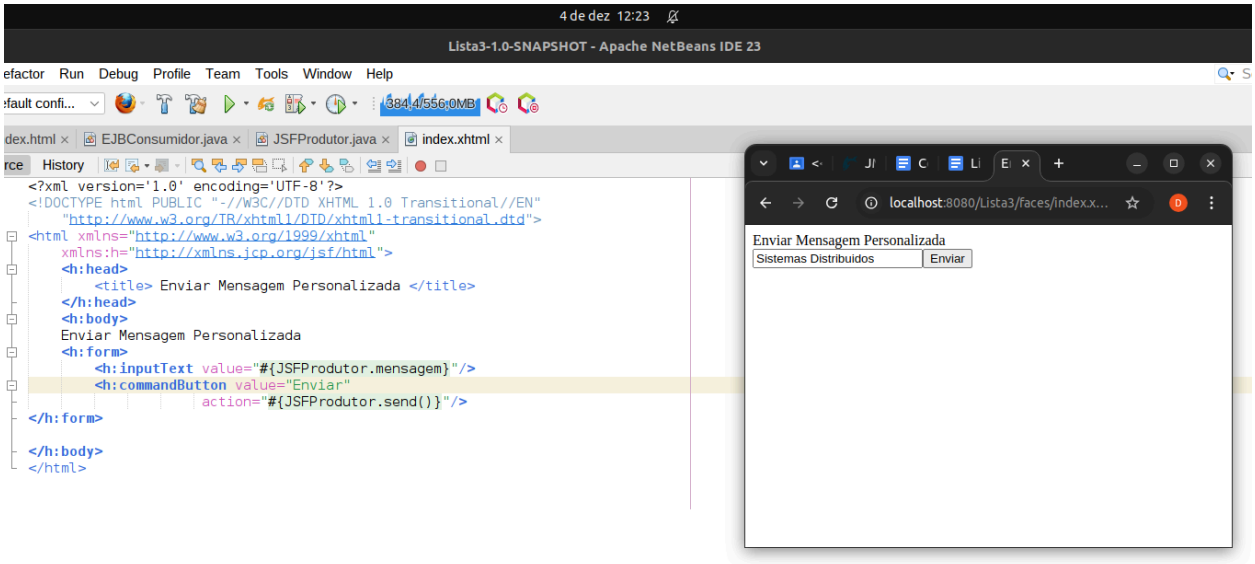
6) O deadlock pode ser detectado por meio da verificação se tem fila circular, ou seja quando há uma transação esperando por um recurso que outra transação está segurando e essa outra transação também está esperando por um recurso da anterior, ficando ambas travadas.
No caso
s1(A) s1(D) r2(B) s1(B) s3(D) s3(C) r2(C) r4(B) r3(A)
t1 tranca (A)
t1 tranca (D)
t2 tranca (B)
t1 vai pra Fila, pois t2 tem a tranca de B
t3 vai para Fila, pois t1 tem a tranca de D
t3 tranca (C)
t2 vai para Fila, pois t3 tem a tranca de C
t4 tranca (B)
t3 vai pra fila, pois t1 tem a tranca de A

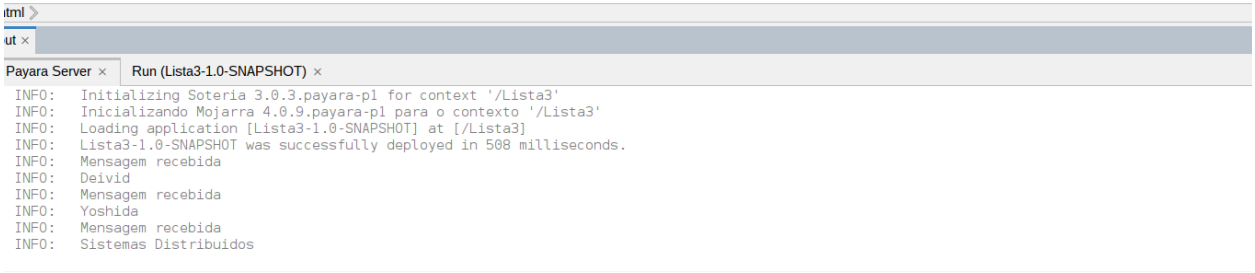
Dependências
t1 → t2 (B)
t3 → t1 (D)
t2 → t3 (C)
t3 → t1 (A)

Logo, é possível verificar que há ocorrência de uma fila circular (t1 → t2 → t3 → t1) e por conta disso pode-se concluir que há deadlock.

7)


Google Chrome
4 de dez 11:48
Apache NetBeans IDE 23
View Navigate Source Refactor Run Debug Profile Team Tools Window Help
localhost:8080/Lista3/faces/index.x...
Enviar Mensagem para o Servidor
Enviar
EJBConsumidor.java
JSFProdutor.java
History
package br.jsf;
import jakarta.annotation.Resource;
import jakarta.inject.Named;
import jakarta.enterprise.context.RequestScoped;
import jakarta.jms.ConnectionFactory;
import jakarta.jms.JMSConnectionFactory;
import jakarta.jms.JMSContext;
import jakarta.jms.Queue;
@author lucio
@Named(value = "JSFProdutor")
@RequestScoped
public class JSFProdutor {
@Resource(lookup="java:comp/DefaultJMSConnectionFactory")
private ConnectionFactory connectionFactory;
@Resource(lookup="java/Fila")
private Queue fila;
/**
* Creates a new instance of JSFProdutor
*/
public JSFProdutor() {
}
public void send(){
try {
JMSContext context = connectionFactory.createContext();
context.createProducer().send(fila, "Ola Mundo");
} catch (Exception e){
br.jsf.JSFProdutor
fila
Output
Payara Server
Run (Lista3-1.0-SNAPSHOT)
INFO: Clustered CDI Event bus initialized
INFO: Initializing Soteria 3.0.3.payara-p1 for context '/Lista3'
INFO: Inicializando Mojarra 4.0.9.payara-p1 para o contexto '/Lista3'
INFO: Loading application [Lista3-1.0-SNAPSHOT] at [/Lista3]
INFO: Lista3-1.0-SNAPSHOT was successfully deployed in 611 milliseconds.
INFO: JTS5014: Recoverable JTS instance, serverId=[100]
INFO: Mensagem recebida
INFO: Mensagem recebida
INFO: Mensagem recebida
INFO: Mensagem recebida





Link do Git

8) Link do Git

* items
* 1) DONE --> src/PeerLista.java /linha 31
* 2) DONE --> src/Peer.java /linha 173
* 3) DONE --> src/Peer.java /linha 206
* 4) DONE --> src/Peer.java /linha 159