Lab 2a – Transações Concorrentes  
   
Problema da Atualização Perdida (10%)

Entrega: 16/09/2013

Este laboratório ilustra dois conhecidos problemas de transações concorrentes no contexto de transações bancárias: o problema da “atualização perdida” e o problema das “recuperações inconsistentes”.

Sejam três contas bancárias A, B e C, de um banco, cujos saldos iniciais são de A=$100, B=$200 e C=$300. Considere duas transações T e U executadas concorrentemente. A transação T transfere um valor da conta A para a conta B. A transação U transfere um valor da conta C para a conta B. Nos dois casos, o valor transferido é calculado de forma a aumentar o saldo de B em 10%. O efeito líquido da execução das transações T e U sobre a conta B deve ser o aumento do seu saldo em 10%, duas vezes; portanto seu valor deverá ser de $242.

Agora considere os efeitos de permitir que as transações T e U sejam executadas concorrentemente como na Figura 13.5 seguinte:

Na Figura 13.5, mostramos as operações que afetam o saldo da conta B em sucessivas linhas e o leitor deve supor que uma operação em uma linha específica é executada depois da que está na linha acima dela.

O problema da atualização perdida ocorre quando duas transações lêem o valor antigo de uma variável e depois o utilizam para calcular o novo valor. Isso não pode acontecer se uma transação for executada antes da outra (em série), pois a transação posterior lerá o valor escrito pela anterior.

* Se for sabido que cada uma de várias transações tem o efeito correto quando executada sozinha ...
* Então, pode-se inferir que, se essas transações forem executadas uma por vez, em alguma ordem, o efeito combinado também será correto.
* Uma intercalação das operações das transações em que o efeito combinado é igual ao que seria se as transações tivessem sido executadas uma por vez, em alguma ordem, é uma intercalação equivalente em série.
* Quando dizemos que duas transações diferentes têm o mesmo efeito, queremos dizer que as operações de leitura retornam os mesmos valores e que as variáveis de instância dos objetos têm os mesmos valores no final.

O uso de equivalência serial como critério para uma execução concorrente correta evita a ocorrência de atualizações perdidas, e recuperações inconsistentes a ser visto no Lab 2b.

Como a intercalação equivalente em série de duas transações produz o mesmo efeito de uma serial, podemos resolver o problema da atualização perdida por meio da equivalência serial.

A Figura 13.7, no que segue, ilustra uma dessas intercalações, na qual as operações que afetam a conta compartilhada B, na verdade, são seriais, pois a transação T executa todas as duas operações sobre B, antes que a transação U o faça. Outra intercalação de T e U que tem essa propriedade é aquela em que a transação U conclui suas operações sobre a conta B, antes que T comece.

Tarefa de Programação de Correção da Atualização Perdida

As transações T e U devem ser programadas de modo que seus efeitos sobre os dados compartilhados sejam equivalentes em série. Um exemplo de mecanismo de sincronização para disposição de transações em série é o uso de travas exclusivas (exclusive locks). Neste esquema, o servidor tenta impedir o acesso (travar) a qualquer variável - por exemplo, a conta B - que esteja para ser usada por qualquer operação na transação de um cliente. Se um cliente solicitar acesso a uma variável – conta B – que já está travado devido à transação de outro cliente, o pedido será suspenso e o cliente deverá esperar até que a variável seja destravada.

A Figura 13.14 (procure ver a figura 4 nos slides), ilustra o uso de travas exclusivas (locks) entre duas transações. Faça um programa que execute o funcionamento das transações como estão representadas na figura 4 dos slides, na qual é mostrado o uso de travas sobre a conta B. Suponha que em toda parte que cada uma das operações *deposit, withdraw, getBalance, setBalance,* seus efeitos sobre a variável de instância que altera o saldo de uma conta são atômicos.

Observações:

1. Antes de fazer o Lab2a, procure ver como funciona o exemplo Produtor-Consumidor do Deitel, em [**Relacionamento Produtor-Consumidor com sincronização usando Locks**](file:///C:\Users\bosco\Documents\ine5645\ine5645\locks.txt) **(Exemplo Deitel 6Ed. Cap. 23, Seção 23.7, pag. 800-805).**
2. Procurem ver o código de Transações, como segue. Este serve para criar as transações. Procure adaptar como achar melhor.
3. Vale peso de 20% na nota da tarefa Lab 2a.
4. Código para uma Transação  
   abstract class Transacao {

int valor;

String data;

Transacao(int valor, String data) {..}

int getValor(){..}

String getData() {..}

abstract int getVariação();

abstract void emiteLinhaExtrato();

}

Transação de crédito

class TransacaoCredito extends Transacao

TransacaoCredito(int valor, String data) {..}

int getVariação() {..}

void emiteLinhaExtrato() {..}

}

Transação de Débito

class TransacaoDebito extends Transacao {

TransacaoDebito(int valor, String data) {..}

int getVariação() {..}

void emiteLinhaExtrato() {..}

}

Template de código: Conta Bancária, precisamos das contas A, B e C

class ContaBancaria {

int numero;

int saldoAnterior;

Transacao[] transacoes;

ContaBancaria(int numero) {..}

int getNumero(){..}

int getSaldoAtual() {..}

void debite(int valor, String data) {..}

void credite(int valor, String data) {..}

void emiteExtrato() {..}

}

Criação de Contas e Transações

class ContaBancaria {

..

public static void main(String[] args) {

ContaBancaria conta =

new ContaBancaria(...);

..

conta.debite(20, “16/09/2013”);

conta.credite(40, “16/09/2013”);

..

conta.emiteExtrato();

..

}