MAC 438 - Programação Concorrente - Primeiro Semestre de 2001

Terceiro Exercício-Programa: `ServContas`

Data de Entrega: 8 de junho de 2001

Este exercício deve ser desenvolvido em equipes de duas pessoas, a fim de suscitar discussão. Dúvidas sobre o enunciado devem ser enviadas para `reverbel-mac438@ime.usp.br`.

1 O problema

Você deve implementar o programa `ServContas`, que gerencia N contas bancárias. As contas são numeradas de 1 a N. Para cada conta são guardadas somente duas informações: o saldo atual e o limite mínimo do saldo.

O programa `ServContas` é um servidor que aceita conexões TCP num port especificado na linha de comando. Conectando-se ao servidor, um cliente pode efetuar as operações saldo, depósito, saque, transferência, info e limite, conforme o exemplo abaixo (esse exemplo supõe que o servidor está aguardando conexões no port 6789 da máquina `algum.host`):

```
$ telnet algum.host 6789
Trying xxx.xxx.xxx.xxx...
Connected to algum.host.
Escape character is '^]'.
Bem vindo!
*saldo 1 4 6
conta 1: 100
conta 4: 1000
conta 6: -2000
*deposito 200 1
conta 1: 100 -> 300
*transf 50 4 1
conta 4: 1000 -> 950
conta 1: 300 -> 350
*saque 100 1
conta 1: 350 -> 250
*saque 1000 6
saque ultrapassaria limite
*info 6
conta 6: saldo -2000, limite -2500
*limite -5000 6
limite da conta 6: -2500 -> -5000
*saque 2000 6
conta 6: -2000 -> -3000
*quit
Volte sempre!
$
```

As três linhas abaixo da chamada `telnet` foram geradas pelo próprio `telnet`. O "`*`" é o prompt do servidor de contas. As linhas iniciadas com "`*`" foram digitadas pelo usuário do `telnet` (exceto pelo primeiro caracter da linha, que foi mandado pelo servidor). Todas as outras linhas foram enviadas pelo servidor. Note que o usuário do telnet pode manipular qualquer conta (pense nele como um funcionário do banco especialmente autorizado para isso).

2 Requisitos da solução

1. O servidor de contas deve ser multithreaded. Para cada conexão TCP recebida, o programa `ServContas` deve criar uma thread que atende o cliente naquela conexão.

   No arquivo `ServidorPrimos.java` você encontrará um exemplo de programa que procede dessa maneira. O programa `ServidorPrimos` é um servidor que aceita uma seqüência de números (através de uma conexão TCP) e, para cada número, devolve ao cliente uma string que informa se o número é primo ou não. Para cada conexão recebida ele cria uma thread para lidar com a seqüência de números oriunda desta conexão. O recebimento de uma linha vazia (uma `String` vazia) indica que o cliente não tem mais números para fornecer ao servidor.

2. As operações saldo, depósito, saque, transferência, info e limite devem ter a propriedade de *isolação*, ou seja: mesmo que várias operações estejam sendo executadas concorrentemente, o resultado de uma dada operação *op* é o que seria obtido caso ela fosse a única em execução no servidor e as operações concorrentes com *op* tivessem rodado antes de *op* (algumas dessas operações) ou depois de *op* (as demais operações). Em outras palavras, a propriedade de isolação exige que o resultado da execução de operações concorrentes seja igual ao resultado de alguma execução serial (numa ordem qualquer) dessas mesmas operações.

   Para deixar isso mais concreto, pense num cliente transferindo X da conta A para a conta B. O servidor pode passar por um estado intermediário, em que X já foi subtraído do saldo de A mas ainda não foi adicionado ao saldo de B, mas esse estado não deve ser visível para nenhum outro cliente! Se, concorrentemente com a transferência, algum cliente pedir (numa só operação) os saldos das duas contas, ele deve receber os dois saldos antes da transferência ou os dois saldos depois da transferência, mas nunca um estado intermediário.

   O programa `ServContas` deve garantir a isolação das operações usando read/write locks (um lock por conta) e o protocolo two-phase locking (2PL) discutido em classe. Sugestão: use o pacote Java `util.concurrent`, disponibilizado por Doug Lea. Esse pacote tem várias implementações de read/write locks.

3. Com o uso de locks, seu servidor pode entrar em deadlock se você não escrevê-lo tomando os devidos cuidados. O `ServContas` deve utilizar uma estratégia de prevenção de deadlocks baseada na ordenação das aquisições de lock: as operações saldo e transferência, que adquirem mais de um lock de conta, devem adquirir os locks sempre numa mesma ordem (em ordem crescente do número da conta, por exemplo).

4. O `ServContas` aceita na linha de comando o argumento opcional "`-cozinh`", que desliga a isolação. Com esse argumento, o servidor ignora o protocolo 2PL e roda em "modo cozinhado", sem adquirir/liberar locks de contas.

5. O `ServContas` aceita na linha de comando o argumento opcional "`-suicida`", que desliga a prevenção de deadlock. Com esse argumento, o servidor adquire os locks de contas na ordem em que as contas aparecem nos operações requisitadas pelos clientes. O comando `saldo 2 3 1` adquiriria o lock da conta 2, depois o da conta 3 e, por último, o da conta 1.

6. Além do programa `ServContas`, você deve implementar um conjunto de clientes que façam seu servidor funcionar mal quando chamado com "`-cozinh`" ou com "`-suicida`".

   - Para o "`-cozinh`", implemente um cliente que fica fazendo sucessivas operações de transferência entre duas contas e outro que repete a operação que dá o saldo dessas duas contas até obter um par de saldos que não deveria ser visto.
   - Para o "`-suicida`", implemente um par de clientes que faz o servidor entrar em deadlock.

7. Como o acesso às contas na memória é muito rápido, pode ser difícil fazer os problemas acima se manifestarem. Para facilitar isso, o `ServContas` pode reconhecer mais um argumento na linha de comando: "`-atraso n`" (onde `n` é um inteiro positivo) faz com que cada acesso (leitura ou escrita) ao

saldo ou ao limite de uma conta leve `n` milisegundos. Se for chamado com esse argumento, o servidor inserirá um `sleep(n)` antes de cada acesso ao saldo ou ao limite de uma conta. Junto com o servidor e com os clientes você deverá incluir um relatório contando se foi ou não foi difícil fazer esses problemas aparecerem, a partir de que valor de `n` eles começam a aparecer, etc.

3 Seu arsenal

Você deve implementar o `ServContas` em Java, usando o JDK 1.2 ou ou o 1.3. Não chame nenhum método "deprecated". Voce pode usar também o pacote `util.concurrent` disponibilizado por Doug Lea.

Os clientes pedidos no item 6 da seção anterior podem ser implementados em Java ou em outra linguagem qualquer, desde que evidenciem os problemas a que o servidor estará sujeito caso seja desligada a isolação ou a prevenção de deadlock. Minha impressão é que o melhor é usar Java também para os clientes, mas fiquem à vontade para discordar de mim!

4 Sobre a entrega

Você deverá entregar três coisas:

- um arquivo tar.gz contendo sua solução (arquivos-fonte, makefile, README, ...);
- uma listagem impressa dos seus arquivos-fonte;
- o relatório sobre a dificuldade que você teve (ou não teve) para evidenciar a ausência de isolação ou conseguir deadlock.

O arquivo `README` deve incuir uma explicação de como rodar os clientes para evidenciar a ausência de isolação num servidor chamado com "`-cozinh`" e para chegar a um deadlock num servidor chamado com "`-suicida`". O que se deseja aqui é uma receita simples para chegar rapidamente a uma situação errônea induzida por clientes escritos com esse propósito. Não é aceitável algo como "rode o cliente C1 e o cliente C2 e espere algumas horas; o deadlock geralmente ocorre depois de quatro horas."

Por favor, entregue o relatório e a listagem **na secretaria do MAC**, em um saco plástico devidamente fechado, contendo também um disquete com o arquvo tar.gz.

Trabalhos atrasados não serão aceitos!

Bom trabalho!