Segundo EE de PLC (IF686)

Data: 10 de dezembro de 2010.

Tempo disponível: 2h00m.

Prof.: Fernando Castor

1. (5,0 ptos., com cada acerto valendo 0,5 pto.) Considere o programa **Java** em a seguir.

```
1  public class C implements Runnable {
2
3    private long f1 = 300000;
4    private long f2 = 300000;
5    private C[] cs = new C[2];
6
7    public void m1(C c1) {
8      C cx = (c1==cs[0]?cs[1]:cs[0]);
9      this.f2 = this.f2 + c1.m2(cx);
10   }
11   public long m2(C cx) {
12     return this.m3() + cx.m3();
13   }
14   public long m3() {
15     this.f1 = this.f1 - 250;
16     this.f2 = this.f2 - 1000;
17     this.f1 = this.f1 - 250;
18     return 500;
19   }
20   public void run() {
21     for(int i = 0; i < 1000000; i++) {
22       cs[1].m1(this); cs[0].m1(cs[1]); this.m1(cs[0]);
23     }
24   }
25   public static void main(String args[]) {
26     C c1 = new C(); C c2 = new C(); C c3 = new C();
27     c1.cs = new C[] {c2,c3};
28     c2.cs = new C[] {c1,c3};
29     c3.cs = new C[] {c1,c2};
30     Thread t1 = new Thread(c1);
31     Thread t2 = new Thread(c2);
32     Thread t3 = new Thread(c3);
33     t1.start(); t2.start(); t3.start();
34     try { t1.join(); t2.join(); t3.join();
35     } catch(InterruptedException e) {}
36   }
37 }
```

Marque verdadeiro (V) ou falso (F) em sua folha de respostas para as proposições a seguir. É importante enfatizar que a pontuação máxima desta questão é de 5,0 pontos. Além disso, **cada dois erros** anulam um acerto. Isso significa que é possível errar um dos itens e, ainda assim, obter nota máxima na questão. Também é possível deixar duas das alternativas em branco e, ainda assim, obter nota máxima na questão.

Obs.1: nas proposições abaixo, sempre que é dito que o programa "produziria a mesma saída que produziria se fosse estritamente sequencial", isso significa que para **todas** as suas possíveis execuções, o resultado será sempre o mesmo e o programa não entra em *deadlock*.
Obs.2: nas proposições abaixo, sempre que é dito que o programa "poderia entrar em *deadlock*", isso significa que esse evento pode acontecer em **algumas** das suas execuções (não necessariamente todas).

(a) Se as linhas 34 e 35 não estiverem presentes, a *thread* principal do programa pode terminar (e, consequentemente, o programa será encerrado) antes que as outras *threads* finalizem sua execução.

(b) Se os métodos `m1()`, `m2()` e `m3()` tivessem o modificador `synchronized`, o programa produziria a mesma saída que produziria se fosse estritamente sequencial (mais especificamente, os atributos `f1` e `f2` de todas as *threads* teriam valor -29700000).

(c) Se os atributos `f1` e `f2` da classe `C` fossem do tipo `AtomicLong` (com operações como soma e subtração sendo implementadas usando os métodos correspondentes definidos por esta classe), o programa produziria a mesma saída que produziria se fosse estritamente sequencial.

(d) Se as linhas 8 e 9 do programa forem substituídas pelas linhas

```
synchronized (cs[0]) {
    synchronized (cs[1]) {
        C4 cx = (c1==cs[0]?cs[1]:cs[0]);
        this.f2 = this.f2 + c1.m2(cx);
    }
}
```

o programa pode entrar em *deadlock*.

(e) Se for incluído no programa o atributo

```
private static C ccc = new C();
```

e a linha 9 for substituída por

```
synchronized(ccc) {
    this.f2 = this.f2 + c1.m2(cx);
}
```

o programa produzirá a mesma saída que produziria se fosse estritamente sequencial.

(f) Suponha que foi incluído no programa o atributo

```
private static Lock[] locks = new Lock[3];
```

Cada posição de `locks` é inicializada no método `main()` com um objeto do tipo `ReentrantLock`. Além disso, suponha que a linha 22 foi substituída pelas seguintes linhas:

```
locks[0].lock();
cs[1].m1(this);
locks[0].unlock();
locks[1].lock();
cs[0].m1(cs[1]);
locks[1].unlock();
locks[2].lock();
this.m1(cs[0]);
locks[2].unlock();
```

Se essas modificações forem feitas, o programa produzirá a mesma saída que produziria se fosse estritamente sequencial.

(g) Agora, levando em conta a letra (g), suponha que a linha 22 foi substituída pelas seguintes linhas:

```
locks[0].lock();
locks[1].lock();
locks[2].lock();
cs[1].m1(this);
cs[0].m1(cs[1]);
this.m1(cs[0]);
locks[1].unlock();
locks[0].unlock();
locks[2].unlock();
```

Se essas modificações forem feitas, o programa produzirá a mesma saída que produziria se fosse estritamente sequencial.

(h) Se a linha 9 do programa for substituída pelas linhas

```
synchronized (cs[0]) {
    this.f2 = this.f2 + c1.m2(cx);
}
```

e a linha 12 for substituída pelas linhas

```
synchronized(cs[1]) {
    return this.m3() + cx.m3();
}
```

o programa pode entrar em *deadlock*.

(i) Se as linhas 27, 28 e 29 do programa forem substituídas pelas linhas

```
c1.cs = new C6[] {c2,c3};
c2.cs = new C6[] {c3,c1};
c3.cs = new C6[] {c1,c2};
```

o programa pode entrar em *deadlock*.

(j) Suponha agora que, ao invés de usar os campos `f1` e `f2`, os valores desses campos sejam sejam armazenados em um `ConcurrentHashMap` cujas chaves sao duas constantes (por exemplo, os `String`s ``um'' e ``dois''. Considere que todas as operações são realizadas obtendo-se o valor atual de cada campo a partir da tabela (usando o método `get()`) e gravando na mesma chave o resultado de cada operação (usando o `put()`). Fora isso, suponha que o programa é funcionalmente equivalente ao original. Se essas modificações forem feitas, o programa produzirá a mesma saída que produziria se fosse estritamente sequencial.

(k) Agora suponha que Java implemente memória transacional. Para este fim, a linguagem poderia incluir um novo tipo de bloco, `atomically`, similar ao `synchronized`, cujo objetivo é fazer com que um determinado trecho de código seja executado dentro de uma transação. Esse bloco poderia ser usado como no exemplo a seguir:

```
atomically {
    return this.m3() + cx.m3();
}
```

Neste cenário, se envolvermos a linha 22 do programa em um bloco `atomically` e se as variáveis `f1` e `f2` passarem a ser de um tipo hipotético `LongTVar`, análogo ao tipo `TVar` de Haskell, o programa produzirá a mesma saída que produziria se fosse estritamente sequencial.

(l) Levando em conta a letra (k), mas considerando o método `run()` original, se os corpos dos métodos `m1()`, `m2()` e `m3()` forem envolvidos por blocos `atomically`, o programa pode entrar em *deadlock*.

2. (5,0) Considere o programa em Haskell a seguir:

```
import System.Random
import Control.Parallel

-- a e b sao matrizes quadradas. Cada lista interna
-- representa uma linha da matriz.
matMult :: [[Int]] -> [[Int]] -> [[Int]]
matMult a b = mm a b

mm :: [[Int]] -> [[Int]] -> [[Int]]
mm [] _ = []
mm _ [] = []
mm (a:as) bs = (multLin a bs 0):mm as bs

multLin :: [Int] -> [[Int]] -> Int -> [Int]
multLin [] _ _ = []
multLin _ [] _ = []
multLin a b i
  | length b == i = []
  | otherwise = (multCol a (pegaCol b i)):multLin a b (i+1)

pegaCol :: [[Int]] -> Int -> [Int]
pegaCol [] _ = []
pegaCol (b:bs) n = (b !! n) : pegaCol bs n

multCol :: [Int] -> [Int] -> Int
multCol [] _ = 0
multCol _ [] = 0
multCol (a:as) (b:bs) = (b * a) + multCol as bs

main :: IO ()
main = do let i = foldr (+) 0 (foldr (++) []
                                (matMult (gm 200) (gm 200)))
          pseq i return()
          putStrLn (show input)

gm :: Int->[[Int]]
... -- gera uma matriz quadrada com nums. aleatorios
```

(a) (2,5 pto.) Modifique o programa acima para que passe a funcionar de forma paralela usando as construções de Haskell para **paralelismo semi-explícito**. Suas modificações devem tornar o programa mais eficiente do que a versão original (estritamente sequencial) quando ele for executado em uma máquina com dois ou mais processadores e você deve explicar sucintamente porque isso acontece, levando em conta os problemas que foram discutidos em sala de aula. Em sua resposta, inclua apenas definições (funções, constantes) e `import`s que tenham sido modificados ou acrescentados com relação ao programa original (por exemplo, se você não modificar a função `pegaCol`, não é necessário inclui-la na resposta).

(b) (2,5 pto.) Modifique o programa original para que passe a funcionar de forma paralela usando as construções de Haskell para a construção de programas **explicitamente concorrentes** que usem várias *threads*. Assim como no item anterior, em sua resposta, inclua apenas definições (funções, constantes) e `import`s que tenham sido modificados ou acrescentados com relação ao programa original.