Ver. A

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Sistemas Distribuídos

1ª Avaliação

26 de Novembro de 2008 Duração: 1h

Nome: _____ **Número:** _____

I

1 Diga o que entende por sincronização de processos em programação concorrente?

2 Explique sucintamente o o que fazem as duas operações sobre semáforos que estudou.

II

O fragmento de código abaixo apresenta uma possível implementação de uma stack de inteiros e uma classe de gestão de tarefas. Para cada uma das afirmações que se seguem identifque a sua veracidade ou falsidade assinalando claramente a caixa de resposta correspondente..

```
1  class Stack {
2
3      private int n = 0, array[N];
4      ReentrantLock rl = new ReentrantLock;
5
6      public void synchronized push(int v) {
7          if (n == N) wait();
8          array[n] = v;
9          n++;
10         notifyAll();
11     }
12
13     public int synchronized pop() {
14         if (n == 0) wait();
15         n--;
16         notifyAll();
17     }
18
19     public int synchronized top() {
20         while (n == 0) Thread.sleep(1000);
21         return array[n];
22     }
23
24     public int synchronized sum() {
25         return total_aux(0);
26     }
27
28     private int synchronized sum_aux(int i) {
29         if (i == n) return 0;
30         return array[i] + sum_aux(i + 1);
31     }
32
33     public void synchronized
34         transfer(int i, int j, int v) {
35
36         while (i >= n || j >= n) wait();
37         array[i] -= v;
38         array[j] += v;
39     }
40
41     public void length() {
42         return n;
43     }
44
45     public int reset() {
46         rl.lock();
47         n = 0;
48         rl.unlock();
49     }
50 }
51
52 class TodoDone {
53
54     Stack todo = null, done = null;
55
56     public TodoDone(Stack todo, Stack done) {
57         this.todo = todo;
58         this.done = done;
59     }
60
61     public void synchronized task(int v) {
62         todo.push(v);
63     }
64
65     public void synchronized execute() {
66         synchronized (done) {
67             synchronized (todo) {
68                 done.push(todo.pop());
69             }
70         }
71     }
72
73     public void undo() {
74         synchronized (todo) {
75             synchronized (done) {
76                 todo.push(done.pop());
77             }
78         }
79     }
80 }
81
82 class Main {
83
84     public static void main() {
85         Stack s1 = new Stack, s2 = new Stack;
86         TodoDone ds = new TodoDone(s1, s2);
87         // ...
88     }
89 }
```

V F **1.** é seguro executar s1.push() e s1.pop() concorrentemente.

V F **2.** s1.push() pode ser executado concorrentemente por múltiplas threads.

V F **3.** s1.push() e s1.reset() podem executar em concorrência.

V F **4.** s1.push() e s2.push() podem executar em concorrência.

V F **5.** s1.push() e s2.pop() podem executar em concorrência.

V F **6.** s1.push() e s1.length() podem executar em concorência.

V F **7.** s1.sum() bloqueia indefidamente.

V F **8.** s1.sum() pode executar sempre completamente.

V F **9.** a invocação concorrente de s1.transfer() por múltiplas threads está sujeita à situação de deadlock.

V F **10.** a invocação de s1.transfer() está sujeita a starvation relativamente aos métodos s1.push() e s1.pop().

V F **11.** a situação de deadlock ocorre quando o progresso de duas threads depende de uma terceira.

V F **12.** as invocações de notifyAll() no método push() e pop() podem ser substituídas por invocações a notify().

V F **13.** a invocação concorrente de s1.push() e de ds.task() é serializada.

V F **14.** a invocação de ds.task() bloqueia indefinidamente.

V F **15.** a invocação de ds.execute() bloqueia indefinidamente.

V F **16.** pode ocorrer deadlock na invocação concorrente de ds.task() e ds.execute().

III

1 Corrija o código apresentado no grupo anterior e avalie a possibilidade de optimizar o controlo de concorrência da classe TodoDone.