

**[22-23] Sistemas Distribuídos**

## Grade Center

Test Statistics: Questionário sobre programação concorrente

## Test Statistics: Questionário sobre programação concorrente

Name	Questionário sobre programação concorrente
Attempt Score	25.49999
Attempts	84 (Total of 91 attempts for this assessment)
Graded Attempts	84
Attempts that Need Grading	0
Instructions	
Alignments	

**Question 1: Multiple Answer****Average Score 4.04761 points**

Considere o seguinte código:

```
boolean r=false;
int j=0;

void a(int t) throws InterruptedException {
    l.lock();
    while (!r || t != j)
        c.await();
    j++;
    System.out.println(t);
    l.unlock();
}

void b() {
    l.lock();
    r = true;
    c.signal();
    l.unlock();
}
```

Assuma que a(0) e a(1) são executadas concorrentemente por dois threads, antes de um terceiro executar b(). O programa:

Correct Answers	Percent Correct	Percent Incorrect
<input checked="" type="checkbox"/> pode imprimir 0 e depois 1	57.142%	42.857%
<input type="checkbox"/> pode imprimir 1 e depois 0	59.523%	40.476%
<input type="checkbox"/> pode imprimir só 1	79.761%	20.238%
<input checked="" type="checkbox"/> pode imprimir só 0	32.142%	67.857%
<input type="checkbox"/> imprime sempre só 0	97.619%	2.38%
<input checked="" type="checkbox"/> pode não imprimir nada	32.142%	67.857%

## Question 2: Multiple Answer

Average Score 1.84523 points

Quando executamos em threads concorrentes e sem quaisquer outras primitivas de sincronização T1 e T2 sobre um objeto compartilhado da classe D, quais são os resultados que podemos obter?

```
class D {
    List<Integer> c;
    ReentrantLock l;

    void add() {
        try { l.lock();
            c.add(1);
            c.add(2);
        } finally { l.unlock(); }
    }

    List<Integer> get() {
        try { l.lock();
            return c;
        } finally { l.unlock(); }
    }
}

class T1 implements Runnable {
    D d;
    public void run() {
        for(Integer i: d.get())
            System.out.println(i);
    }
}

class T2 implements Runnable {
    D d;
    public void run() {
```

```

        d.add();
    }
}

```

	Correct Answers	Percent Correct	Percent Incorrect
	apenas 1, 2	85.714%	14.285%
	apenas 1,2 ou nada	39.285%	60.714%
✓	1 ou 1,2	16.666%	83.333%
✓	uma exceção	20.238%	79.761%

### Question 3: Multiple Answer

Average Score 5.29761 points

Considere o código do slide 29 (locks) e tenha atenção aos locks do jogo (l) e do jogadores (s.l e t.l):

```

void shoot(String sn, String tn) {
    l.lock();
    Player s = players.get(sn);
    Player t = players.get(tn);
    Stream.of(sn,tn).sorted()
        .forEach(n -> players.get(n).l.lock());
    l.unlock();
    t.life--;
    t.l.unlock();
    s.score++;
    s.l.unlock();
}

```

Sem assumir nada quanto a resto do código que usa os mesmos locks:

	Correct Answers	Percent Correct	Percent Incorrect
✓	é mais eficiente que o lock do jogo (l) seja um ReadWriteLock e o lock do jogador (s.l e t.l) seja um ReentrantLock	64.285%	35.714%
	é necessário que sejam todos ReentrantLocks	95.238%	4.761%
✓	quando o lock do jogo (l) é um ReentrantLock não é necessário ordenar a aquisição dos locks dos jogadores (s.l e t.l)	41.666%	58.333%
	quando os locks dos jogadores (s.l e t.l) são ReadWriteLocks não é necessário ordenar a sua aquisição	73.809%	26.19%

é mais eficiente que sejam todos ReadWriteLocks	86.904%	13.095%
--	---------	---------

#### Question 4: Multiple Answer

Average Score 6.30952 points

Considere o desempenho (*performance*) de programas concorrentes que usam primitivas de sincronização para delimitar e controlar o acesso a secções críticas.

Correct	Answers	Percent Correct	Percent Incorrect
<input checked="" type="checkbox"/>	O desempenho depende da probabilidade de diferentes threads tentarem usar a mesma secção crítica ao mesmo tempo	67.857%	32.142%
	O desempenho depende da apenas do número de threads que são iniciados ao mesmo tempo	96.428%	3.571%
<input checked="" type="checkbox"/>	A utilização de um R/W lock é vantajosa quando o número de acessos à secção crítica de leitura é muito maior que o número de acessos para escrita	71.428%	28.571%
	A utilização de um R/W lock em vez de um mutex (e.g., ReentrantLock) é mais vantajosa quanto mais pequena for a duração da secção crítica para os leitores	83.333%	16.666%

#### Question 5: Multiple Answer

Average Score 0.5 points

Considere o seguinte código-fonte que simula o comportamento de uma máquina de café. A máquina tem uma capacidade máxima de 100 unidades, e disponibiliza dois métodos:

- **tirarCafe()**: remove da máquina uma unidade de café, bloqueando se esta estiver vazia;
- **reabasterMaquina()**: abastece a máquina com 20 unidades de café.

Assuma que a máquina nunca deve ultrapassar o limite da sua capacidade. Considere também que múltiplas threads podem invocar o método **tirarCafe** mas apenas uma thread pode invocar o método **reabastecerMaquina**.

```
int quantidade = 0;  
int capacidade = 100;  
Lock lock = new ReentrantLock();  
Condition cond1 = lock.newCondition();
```


Condition **cond2** = **lock**.newCondition();

```
void tirarCafe () { void reabastecerMaquina () {  
    lock.lock();    lock.lock();  
    (1)_____    (3)_____  
    quantidade--;   quantidade += 20;  
    (2)_____    (4)_____  
    lock.unlock();  lock.unlock();  
} } }
```

Correct	Answers	Percent Correct	Percent Incorrect
<input checked="" type="checkbox"/>	(1) while (quantidade<=0) { cond1.await(); } (2) cond2.signal(); (3) while (quantidade > capacidade - 20) { cond2.await(); } (4) cond1.signalAll();	50.00%	50.00%
	(1) while (quantidade==0) { cond1.await();} (2) if (quantidade <= 20) { cond2.signalAll(); } (3) cond2.await(); (4) cond1.signalAll();	91.666%	8.333%
	(1) while (quantidade > 0) { cond1.await(); } (2) if (quantidade==0) { cond2.signalAll(); } (3) while (quantidade + 20 > capacidade) { cond2.await(); } (4) cond1.signalAll();	77.38%	22.619%
	(1) while (quantidade>0) { cond2.await(); } (2) cond2.signalAll(); (3) while (quantidade + 20 > capacidade) { cond1.await(); } (4) cond1.signalAll();	82.142%	17.857%

Considerando que x é uma variável partilhada, quando executamos o seguinte código em 4 threads ao mesmo tempo, o programa imprime 4 valores de x. Que resultados podemos obter?

```
public void run() {  
    for(int i=0; i<1000; i++)  
        x = x+1;  
    System.out.println(x);  
}
```

Correct	Answers	Percent Correct	Percent Incorrect
	os quatro valores de x impressos são sempre todos iguais a 4000	96.428%	3.571%
	pelo menos um dos valores de x impresso é sempre 4000	80.952%	19.047%
	podemos observar valores de x superiores a 4000	85.714%	14.285%
	podemos observar todos os valores de x inferiores a 4000	75.00%	25.00%

← OK