UFCG UASC/CEEI

Disciplina: Programação Concorrente Prova 2
Aluno:
Matrícula:

1. Considere os algoritmos LockOne e LockTwo. Indique exemplos de execução, para cada um dos algoritmos, em que são quebrados requisitos de segurança e/ou progresso. Explique quais os problemas que acontecem em cada caso. Use a seguinte notação para indicar os exemplos de execução (T0_5 -> T0_6 -> T1_5). Essa notação indica que a thread T0 executou as linhas 5 e 6 e depois a thread T1 executou a linha 5.
2. Explique como o algoritmo de Peterson garante que os problemas apresentados nos algoritmos LockOne e LockTwo são resolvidos.
3. Explique como travas TTAS podem ter desempenho melhor que travas TAS. Sua explicação deve considerar aspectos de arquitetura de computadores.
4. Considere a implementação BrokenBakery. Que problemas, de progresso e segurança, essa implementação incorreta do algoritmo apresenta. Use a mesma notação indicada na questão 1.
5. Refatore a implementação do algoritmo Bakery usando AtomicInt e AtomicBoolean. Simplifique o código ao máximo.

Algoritmos

```
1  class LockOne implements Lock {
2    private boolean[] flag = new boolean[2];
3    // thread-local index, 0 or 1
4    public void lock() {
5      int i = ThreadID.get();
6      int j = 1 - i;
7      flag[i] = true;
8      while (flag[j]) {}        // wait until flag[j] == false
9    }
10   public void unlock() {
11     int i = ThreadID.get();
12     flag[i] = false;
13   }
14 }
```

```
class LockTwo implements Lock {
  private int victim;
  public void lock() {
    int i = ThreadID.get();
    victim = i;               // let the other go first
    while (victim == i) {}    // wait
  }
  public void unlock() {}
}
```

```
class Peterson implements Lock {
  // thread-local index, 0 or 1
  private boolean[] flag = new boolean[2];
  private int victim;
  public void lock() {
    int i = ThreadID.get();
    int j = 1 - i;
    flag[i] = true;            // I'm interested
    victim = i;                // you go first
    while (flag[j] && victim == i) {} // wait
  }
  public void unlock() {
    int i = ThreadID.get();
    flag[i] = false;           // I'm not interested
  }
}
```

```
class BrokenBakery implements Lock {

    volatile int[] ticket;

    public Bakery (int n) {
        ticket = new int[n];
        for (int i = 0; i < n; i++) {
            ticket[i] = 0;
        }
    }

    public void lock() {
        int id = Thread.getID()
        for (int j = 0; j < n; j++)
            if (ticket[j] > ticket[id])
                ticket[id] = ticket[j];
        ticket[id]++;


        for (int j = 0; j < n; j++) {
            while ((ticket[j] != 0) && ((ticket[j] < ticket[id]));
        }
    }

    public void unlock() {
            int id = Thread.getID()
            ticket[id] = 0;
    }
}
```

```
class Bakery implements Lock {
    volatile int[] ticket;
    volatile boolean[] escolhendo;

    public Bakery (int n) {
        ticket = new int[n];
        for (int i = 0; i < n; i++) {
            ticket[i] = 0;
        }
    }

    public void lock(int id) {
        escolhendo[id] = true;
        for (int j = 0; j < n; j++) {
            if (ticket[j] > ticket[id])
                ticket[id] = ticket[j];
        }
        ticket[id]++;
        escolhendo[id] = false;

        for (int j = 0; j < n; j++) {
            while (escolhendo[j]);
            while ((ticket[j] != 0) &&
                ((ticket[j] < ticket[id]) ||
                ((ticket[j] == ticket[id]) && j < id)));
        }
    }

    public void unlock(int id) {
        ticket[id] = 0;
    }

}
```

AtomicBoolean()
Creates a new AtomicBoolean with initial value false.

AtomicBoolean(boolean initialValue)
Creates a new AtomicBoolean with the given initial value.

boolean	**compareAndSet(boolean expect, boolean update)** Atomically sets the value to the given updated value if the current value == the expected value.
boolean	**get()** Returns the current value.
boolean	**getAndSet(boolean newValue)** Atomically sets to the given value and returns the previous value.
void	**set(boolean newValue)** Unconditionally sets to the given value.