# Parte II - Exame Sistemas Operativos 2021/2022

---

## Questão 7 (a) [2.0 valores]

### Pergunta:

Existe a possibilidade de surgirem resultados diferentes ou ocorrer uma situação de impasse (deadlock)?

### Resposta:

Sim, é possível ocorrer \*\*deadlock\*\*.

### Justificação:

No código fornecido, temos duas threads (Alice e Bob) que utilizam os objetos `p1` e `p2` para sincronização.  
  
1. A thread \*\*Alice\*\* bloqueia primeiro `p1` e tenta bloquear depois `p2`.  
2. Ao mesmo tempo, a thread \*\*Bob\*\* bloqueia primeiro `p2` e tenta bloquear depois `p1`.  
  
Se \*\*Alice\*\* obtiver o bloqueio em `p1` e \*\*Bob\*\* obtiver o bloqueio em `p2`, ambas as threads ficarão à espera do recurso bloqueado pela outra, resultando em \*\*deadlock\*\*.

## Questão 7 (b) [2.0 valores]

### Requisitos:

1. Eliminar situações de \*\*deadlock\*\*.  
2. Garantir, com \*\*semáforos\*\*, que a linha 42 só é executada após todas as threads terminarem.

### Código Corrigido:

import java.util.concurrent.Semaphore;  
  
class Normal {  
 private int nota;  
  
 Normal(int nota) { this.nota = nota; }  
 public int getNota() { return nota; }  
}  
  
class ParteTeorica extends Normal {  
 ParteTeorica(int nota) { super(nota); }  
}  
  
class PartePratica extends Normal {  
 PartePratica(int nota) { super(nota); }  
}  
  
class Aluno extends Thread {  
 private String nome;  
 private Normal p1, p2;  
 private Semaphore semaforo;  
  
 Aluno(String nome, Normal p1, Normal p2, Semaphore semaforo) {  
 this.nome = nome;  
 this.p1 = p1;  
 this.p2 = p2;  
 this.semaforo = semaforo;  
 }  
  
 public void run() {  
 synchronized(p1) {  
 System.out.println(nome + " obteve " + p1.getNota());  
 try {  
 if (p1.getNota() < 10) Thread.sleep(1000);  
 } catch (InterruptedException e) { }  
  
 synchronized(p2) {  
 System.out.println(nome + " obteve " + p2.getNota());  
 }  
 }  
 semaforo.release();  
 }  
}  
  
public class ExameEpocaNormal {  
 public static void main(String[] args) {  
 final Normal p1 = new ParteTeorica(15);  
 final Normal p2 = new PartePratica(9);  
  
 Semaphore semaforo = new Semaphore(0);  
  
 Thread t1 = new Aluno("Grupo 1: Alice", p1, p2, semaforo);  
 Thread t2 = new Aluno("Grupo 1: Bob", p2, p1, semaforo);  
  
 t1.start();  
 t2.start();  
  
 try {  
 semaforo.acquire(2);  
 } catch (InterruptedException e) { }  
  
 System.out.println("A nota dos alunos do Grupo 1 foi registada");  
 }  
}

## Questão 8 [4.0 valores]

### Requisitos:

1. Implementar comunicação entre \*\*Professor\*\* e \*\*Estudantes\*\* via um buffer compartilhado (`Email`).  
2. Respeitar a sincronização entre threads e garantir a execução em ciclos.

### Código Resolvido:

import java.util.concurrent.\*;  
import java.util.\*;  
  
class Email {  
 private Queue<String> mensagens = new LinkedList<>();  
 private int capacidade = 5;  
  
 private Semaphore podeEscrever = new Semaphore(1);  
 private Semaphore podeLer = new Semaphore(0);  
  
 public void escrever(List<String> msgs) throws InterruptedException {  
 podeEscrever.acquire();  
 mensagens.addAll(msgs);  
 System.out.println("Professor escreveu: " + msgs);  
 podeLer.release(10);  
 }  
  
 public List<String> ler() throws InterruptedException {  
 podeLer.acquire();  
 List<String> lidas = new ArrayList<>(mensagens);  
 mensagens.clear();  
 if (podeLer.availablePermits() == 0) {  
 podeEscrever.release();  
 }  
 return lidas;  
 }  
}  
  
class Professor extends Thread {  
 private Email email;  
  
 Professor(Email email) { this.email = email; }  
  
 public void run() {  
 for (int i = 0; i < 7; i++) {  
 try {  
 List<String> msgs = Arrays.asList("Msg1", "Msg2", "Msg3", "Msg4", "Msg5");  
 email.escrever(msgs);  
 Thread.sleep(1000);  
 } catch (InterruptedException e) { }  
 }  
 }  
}  
  
class Estudante extends Thread {  
 private Email email;  
  
 Estudante(Email email) { this.email = email; }  
  
 public void run() {  
 try {  
 List<String> mensagens = email.ler();  
 System.out.println(Thread.currentThread().getName() + " leu: " + mensagens);  
 } catch (InterruptedException e) { }  
 }  
}  
  
public class Comunicacao {  
 public static void main(String[] args) {  
 Email email = new Email();  
  
 Professor professor = new Professor(email);  
 professor.start();  
  
 for (int i = 0; i < 10; i++) {  
 new Estudante(email).start();  
 }  
 }  
}