Nome:

. . . . . .

Número:

☐0 ☐0 ☐0 ☐0 ☐0
☐1 ☐1 ☐1 ☐1 ☐1
☐2 ☐2 ☐2 ☐2 ☐2
☐3 ☐3 ☐3 ☐3 ☐3
☐4 ☐4 ☐4 ☐4 ☐4
☐5 ☐5 ☐5 ☐5 ☐5
☐6 ☐6 ☐6 ☐6 ☐6
☐7 ☐7 ☐7 ☐7 ☐7
☐8 ☐8 ☐8 ☐8 ☐8
☐9 ☐9 ☐9 ☐9 ☐9

ENGENHARIA INFORMÁTICA – UNIVERSIDADE DO MINHO

Teste de Sistemas Distribuídos

2 de janeiro de 2023 – Duração: 2h00

Instruções: *Preencha o nome e o número de aluno nesta folha pintando completamente as caixas correspondentes a cada algarismo; em cada pergunta de escolha múltipla há sempre uma ou mais respostas certas; para as assinalar pinte completamente as caixas correspondentes; não use as áreas sombreadas; preencha também o nome e número em cada folha de exame adicional.*

Grupo I

Responda a este grupo no próprio enunciado.

1. O fragmento de código seguinte é executado concorrentemente sem primitivas de exclusão mútua, uma vez por cada um de 100 *threads*.
Assuma que os valores iniciais das variáveis são `a=10` e `b=0`. Os valores finais das variáveis são:

```
if (a > 0) {
  a--; b++;
}
```

- ☐ `a` pode ser menor que 0
- ☐ `b` é sempre menor ou igual a 10
- ☐ `a+b` é sempre igual a 10
- ☐ `b` é sempre maior que 0

2. Considere um sistema cliente/servidor de uma imobiliária que permite efetuar buscas de acordo com vários critérios: tipologia (T1, T2, ...), garagem (S/N), dimensão (m^2), etc. As operações oferecidas na interface deste servidor devem incluir:

- ☐ pesquisa, que recebe um mapa de critérios para valores e devolve uma lista de imóveis encontrados
- ☐ pesquisa, que recebe uma *array* de valores desejados para todos os critérios e devolve um *array* de booleanos indicando quais os imóveis relevantes
- ☐ operações para seleccionar cada um dos critérios e uma outra operação de pesquisa que usa os critérios previamente seleccionados e devolve uma lista de imóveis encontrados
- ☐ envio do número da linha seleccionada num menu, por exemplo, para seleccionar um critério, e devolve o novo conteúdo a afixar no écran

3. Considere um modelo de concorrência *1-thread-por-pedido* na implementação do servidor num sistema cliente/servidor usando *sockets* TCP/IP. É verdade que este modelo:

- ☐ não dispensa a utilização de primitivas de exclusão mútua na lógica da aplicação com estado partilhado do servidor
- ☐ dispensa a utilização de primitivas de exclusão mútua na lógica da aplicação no cliente
- ☐ dispensa a utilização de primitivas de exclusão mútua na manipulação do estado de sessão no servidor
- ☐ é o mais eficiente em recursos do servidor com clientes multi-thread

4. Considere a implementação de um *mutex* distribuído usando um algoritmo de coordenador centralizado. Considerando os algoritmos estudados, é verdade que com 10 processos:

- ☐ a latência média de entrada na secção crítica é menor do que com um algoritmo em anel
- ☐ é a melhor opção em termos largura de banda necessária quando só dois processos usam a secção crítica
- ☐ a latência média de entrada na secção crítica é menor do que com um algoritmo baseado em relógios lógicos (Ricart-Agrawala)
- ☐ é a única alternativa que tolera o *crash* de qualquer processo

5. Considere um sistema replicado em que se pretende mater várias cópias dos mesmos dados, geograficamente distribuídos na Internet e sujeitos a alterações, para tolerar a falha de um centro de dados. Discuta a utilidade e limitações de um algoritmo de escolha de um líder (*bully*) neste contexto. Utilize exemplos.

☐0 ☐.1 ☐.2 ☐.3 ☐.4 ☐.5 ☐.6 ☐.7 ☐.8 ☐.9 ☐1 *correção*

Grupo II *Responda a cada pergunta deste grupo numa folha de exame separada.*

Considere um sistema de *caching* que permite associar chaves (números inteiros) a valores (*byte arrays* de tamanho variável). Em cada momento, podem estar armazenados no máximo `N` chaves. Este sistema disponibiliza as seguintes operações: inserir ou modificar o valor associado a uma chave, que termina sempre com sucesso mas pode bloquear para impedir que o número de pares armazenados ultrapasse `N`; ler o valor associado a uma chave, devolvendo `null` se ela não existir; eliminar uma chave e o valor a ela associado, libertando espaço.

6. Apresente uma classe Java (para ser usada no servidor) que implemente a interface pretendida, tendo em conta que os seus métodos serão invocados num ambiente *multi-threaded*.

```
interface Cache {
  void put(int key, byte[] value);
  byte[] get(int key);
  void evict(int key);
}
```

Valorização: Assegure que escritas na mesma chave são efetuadas por ordem de chegada. Minimize os *threads* que são acordados.

7. Considere um serviço ao qual clientes se ligam por TCP para lerem e escreverem pares na *cache*. Implemente só o programa servidor usando *threads*, *sockets* TCP e a interface apresentada na pergunta anterior.

Valorização: Garanta que uma chave escrita por um cliente e que não é lida nas `M` operações seguintes do mesmo cliente, é eliminada da *cache*.