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

Exame de Sistemas Distribuídos

23 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=b=0`. No final da execução:

```
while (a > 0){}
a=1; b++; a=0;
```

- ☐ `b` pode ser maior que 100
- ☐ `b` pode ser menor que 100
- ☐ `a` pode ser igual a 1
- ☐ `b` pode ser igual a 100

2. No sentido de obter um sistema de ficheiros cliente/servidor robusto e escalável, as operações oferecidas pela interface do servidor devem incluir operações de escrita com os seguintes parâmetros:

- ☐ um identificador do ficheiro alvo, um *array* de *bytes* com o conteúdo e uma posição no ficheiro onde escrever
- ☐ um identificador do ficheiro alvo e um *array* de *bytes* com o conteúdo, escrevendo na posição a seguir à escrita anterior
- ☐ um identificador do ficheiro alvo, um apontador para o conteúdo e uma posição no ficheiro onde escrever
- ☐ um indentificador do ficheiro alvo e o nome do ficheiro local com o conteúdo a carregar

3. Num sistema replicado para tolerância a faltas são mantidas várias cópias dos mesmos dados modificados por operações efetuadas em diferentes servidores. Sem fazer pressupostos quanto ao tempo, é verdade que:

- ☐ um algoritmo de quorum maioritário ($n_R = n_W > n/2$) garante que é lido o último valor escrito
- ☐ ao contrário do algoritmo de quorum, o algoritmo de *2-phase commit* (2PC) nunca bloqueia
- ☐ um algoritmo de quorum em que $n_R = 1$ não tolera faltas por *crash* de processos
- ☐ a eleição de lider com o algoritmo de *bully* permite que haja sempre exatamente um processo responsável pela ordenação

4. Um programa distribuído etiqueta eventos usando um relógio lógico. Registam-se eventos com as seguintes etiquetas: $(a, 4), (b, 5), (c, 8), (d, 4)$. Podemos daqui concluir que:

- ☐ a e d aconteceram no mesmo processo
- ☐ sabendo que a e b aconteceram no processo i, então a aconteceu antes de b (em tempo real)
- ☐ sabendo que b corresponde ao envio de uma mensagem m e d corresponde a recepção de uma mensagem m', é possível que m e m' sejam a mesma
- ☐ c não pode ter acontecido antes de b (em tempo real)

5. Considere um sistema de música na Internet como o Spotify, em que cada utilizador pode selecionar, descarregar e ouvir músicas. Proponha uma arquitetura para este sistema, identifique o problema principal de sistemas distribuídos que ele coloca e discuta as soluções mais apropriadas para o resolver.

☐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 controlo de corridas de Karts. Cada corrida envolve N Karts, começando quando os N participantes estão prontos. Cada participante tem que previamente reservar um dos N Karts existentes, sendo identificado na corrida pelo número do Kart, de 0 a $N-1$. O vencedor é o primeiro participante a completar V voltas. Os Karts começam livres e retornam ao estado livre à medida que os participantes completam a sua volta em curso depois de o vencedor terminar.

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

```
interface Controlador {
  int reserva();
  void preparado(int kart);
  void completaVolta(int kart);
  int[] voltasCompletas();
  int vencedor();
}
```

O método `reserva` devolve o número do Kart livre reservado; `preparado` serve para um participante indicar que está pronto, devendo bloquear até todos os participantes o estarem, começando a corrida nesse momento; `completaVolta` serve para um participante informar que finalizou uma volta à pista; `voltasCompletas` devolve o número de voltas que cada Kart já completou; `vencedor` deve bloquear até a corrida terminar, devolvendo o vencedor.

Valorização: Permita que o controlador seja usado para mais do que uma corrida, fazendo com que o método `reserva` bloqueie até haver um Kart livre. Minimize os *threads* que são bloqueados ou acordados sem necessidade.

7. Considere um serviço ao qual clientes se ligam por TCP para participarem em corridas. Implemente só o programa servidor usando *threads*, *sockets* TCP e a interface apresentada na pergunta anterior implementada de forma a suportar corridas sucessivas. Cada cliente liga-se para participar numa única corrida. Durante a corrida deve poder informar que completou mais uma volta ou pedir para saber quantas voltas cada participante já completou.

Valorização: O cliente deve ser informado imediatamente quando a corrida acaba e quem foi o vencedor (e não apenas depois de completar mais uma volta).