Computação Distribuída

Verdadeiros/Falsos

• No modelo de programação MultiThread, cada uma das Threads executa as instruções de forma sequencial. **Verdadeiro**
• Uma Thread termina se o método run executar todas as instruções. **Verdadeiro**
• Todos os algoritmos sequenciais podem ser paralelizados. **Falso** - **Explicação**: Nem todos os algoritmos sequenciais podem ser paralelizados de forma eficiente. Alguns algortimos têm dependências de dados ou de estado que os impedem de ser executados eficientemente em parelelo
• No modelo de execução MultiThread não é possível saber a ordem pela qual as instruções são executadas. **Verdadeiro**
• Os métodos das estruturas de dados para processamento concorrente são mais rápidos que os seus equivalentes nas estruturas para processamento sequencial. **Falso** - **Explicação** - Os métodos concorrentes não são necessariamente mais rápidos que os sequenciais. Introduzem overhead de sincronização e condições de corrida, podendo ser mais lentos dependendo da aplicação.
• A instrução exclusive permite fazer o acesso exclusivo a um método de um objeto. **Falso** - **Explicação** - A instrução exclusive não garante acesso exclusivo a um método de um objeto. Ela apenas impede que mais de uma thread execute aquele bocado de código ao mesmo tempo. Outras threads ainda podem acessar o mesmo método concorrentemente. A instrução **syncronized** permite.
• Um servidor não bloqueante é necessariamente MultiThread. **Verdadeiro**
• Os sockets permitem que aplicações distintas comuniquem dentro do mesmo computador. **Verdadeiro**
• Os sockets TCP possuem Timeout que permite que as aplicações não fiquem bloqueadas á espera de mensagens que não chegam. **Verdadeiro**
• A comunicação com Sockets TCP é feita com objetos da classe DatagramPacket. **Falso** - **Explicação** - A comunicação com sockets TCP utiliza objetos da classe Socket e os fluxos de entrada/saída (InputStream e OutputStream) associados, não à classe DatagramPacket. Mas sim comunicação orientada por sockets via **UCP**
• RPC é a abreviatura de Remote Process Call. **Verdadeiro**
• Os sockets TCP são uma implementação da tecnologia RPC. **Falso** - **Explicação** - Socket TCP lidam apenas coma troca confiável de fluxo de bytes, enquanto RPC requer semântica de mais alto nível para chamada de funções remotas com parâmetros.
• Na tecnologia RMI os clientes utilizam referências para os objetos remotos. **Verdadeiro**
• Em Java as chaves do algoritmo AES são instâncias da classe KeyPair. **Falso** - **Explicação** - KeyPair está associada à criptografia assimétrica, enquanto as chaves AES são chaves simétricas representadas por SecretKey.
• O algoritmo SHA permite encriptar uma mesagem. **Falso** - **Explicação** - O SHA é um algoritmo de hast criptográfico, que gera uma representação compacta de tamanho fixo de qualquer mensagem dado. A função hash é unidirecional, não permitindo a recuperação da mensagem original.
• A tecnologia blockchain foi desenvolvida para o armazenamento centralizado de factos. **Falso** - **Explicação** - A tecnologia blockchain foi desenvolvida para armazenamento descentralizado e

distribuído de dados/transações, e não é centralizado.
• A validação de um bloco dentro de uma blockchain é feita com assinaturas digitais. **Falso** - **Explicação** - As assinaturas digitais comprovam que quem fez a transação está ciente do contúdo ali adicionado. Sendo assim, quandoa esse dado é colocado na blockchain, é gerado uma validação digital que o identifica e bloqueia a alteração do seu conteúdo, essa validação é feita pelos códigos hash.
• A validação de um bloco dentro de uma blockchain é um processo moroso. **Falso** - **Explicação** - A validação de um bloco não é necessariamente morosa, mas a criação desse bloco através da mineração é que pode ser demorada e competitiva.
• Os objetos de uma classe que implemente a interface Runnable podem ser parâmetro do construtor da classe Thread. **Verdadeiro**
• Quando se chama o método start de uma Thread o método run é executado imediatamente. **Falso** - **Explicação** - Antes de executar o método run a Thread executa o método start primeiro que inicia uma nova thread e em seguida o método run.
• Uma classe que herde a classe Thread pode ser executada de forma assíncrona chamando o método run - **Falso** - **Explicação** - O método run é chamado de forma síncrona e o start de forma assíncrona.
• Todos os algoritmos aumentam o seu desempenho em computadores multi-core através da sua paralelização - **Falso** - **Explicação** - Alguns algortimos são intrisecamente sequenciais e não podem ser facilmente divididos em tarefas independentes que podem ser executadas em paralelo. Em casos como esse, não é possível obter ganhos significativos de desempenho em um ambiente multi-core, a menos que o algoritmo seja completamente reformulado.
• Todos os programas têm pelo menos uma Thread. **Verdadeiro** - O main() é uma Thread.
• Em Java, o acesso exclusivo de uma Thread a um método é feito pela palavra reservada exclusive. **Falso** - **Explicação** - Para este efeito usa-se syncronized ou lock,
• O algortimo RSA é utilizado para fazer criptografia simétrica. **Falso** - Algoritmos de criptografia simétrica: DES, 3DES, IDEA, AES
• Os algoritmos de criptografia assimétrica são mais rápidos que os simétricos - **Falso** - Algoritmos de criptografia simétrica usam uma única chave para criptografar e descriptografar dados, o que é um processo mais eficiente em termos de recursos computacionais.
• No modelo de execução concorrente não é possível saber a ordem pela qual as instruções são executadas. **Verdadeiro**
• A aceleração máxima de auma algoritmo paralelo é limitada pelo número de unidades de processamento disponíveis. **Verdadeiro**
• Os métodos das estruturas de dados para processamento concorrente são mais rápidos que os equivalentes nas estruturas para processamento sequencial. **Falso** - **Explicação** - A eficiência depende de fatores como o contexto de uso, a implementação específica e os padrões de acesso aos dados. Estruturas de dados concorrentes geralmente introduzem mecanismos de sincronização, que podem adicionar sobrecarga, tornano operações concorrentes mais lentas que as operações sequenciais.
• A interface Callable executa cálculos de forma síncrona. **Falso** - A interface Callable é usada em conjunto com a classe ExecutorService para executar cálculos de forma assíncrona.
• Um processo termina quando a thread main terminar. **Falso** - O processo termina quando todas as threads terminarem.

• Em Java, uma Thread adormecida pelo método Sleep é acordada pelo método notify - **Falso** - **Explicação** - Com o método sleep, não se usa nenhum método de notificação porque este método tem um parâmetro para meter o tempo desejado que a Thread esteja adormecida e só depois desse tempo passar é que ela "acorda".
• A criptografia simétrica não garante a identidade de quem enviou a mensagem. **Verdadeiro** - Criptografia simétrica utiliza apenas uma chave tanto para criptografar como para descriptografar, por isso se mais que um utilizador tenha posse dessa chave pode descriptografar esses dados, assim não conseguimos saber quem o fez.
• O algoritmo MD5 permite verificar a integridade de uma mensagem. **Verdadeiro** - O MD5 é usado para calcular um hash fixo de 128 bits a partir de uma mensagem ou arquivo, qualquer alteração na mensagem resulta num hash diferente, assim conseguimos saber se a mensagem foi alterada.
• No modelo de programação MultiThread, cada uma das Threads executa as instruções de forma sequencial. **Verdadeiro**
• Os programas paralelos fornecem sempre o mesmo output quando é introduzido o mesmo input. **Falso** - **Explicação** - Isto pode acontecer por causa de recursos compartilhados, pela ordem das instruções, má sincronização.
• Um objeto do tipo SwingWorker executa as do método doInBackground() de forma sequencial. **Falso** - **Explicação** - O objetivo principal do SwingWorker é executar tarefas em segundo plano de forma assíncrona, para evitar bloquear a interface do utilizador durante a execução das tarefas.
• Todos os algoritmos podem ser paralelizados. **Falso** - **Explicação** - No caso do algoritmo depender do resultado da operação anterior não é possível paralelizar o algoritmo.
• Duas Threads podem partilhar a mesma CPU. **Verdadeiro**
• A execução de uma Thread é mais complexa do que a execução de um processo. **Falso** - **Explicação** - Processos são unidades independentes com o seu próprio espaço de endereçamento, recursos do sistema separados e trocas de contexto mais custosas.
• Todos os programas terminam quando executarem todas as instruções da função principal (main). **Falso** - **Explicação** - O processo termina quando todas as threads terminarem.
• Os programas paralelos fornecem sempre o mesmo output quando é introduzido o mesmo input. **Falso**
• Duas Threads podem ter acesso a um objeto em memória. **Verdadeiro**
• A instrução synchronized permite bloquear blocos de código dentro de métodos. **Verdadeiro**
• O deadlock é um erro que ocorre apenas em ambientes de execução com várias threads. **Verdadeiro** - Se apenas houver uma Thread não existe concorrência de recursos, ou seja, não existe deadlock.
• Uma Thread em java é executada de forma assíncrona chamando o método run. **Falso** - **Explicação** - O método start sim permite a execução assíncrona.
• Um método synchronized impede que outro synchcronized seja executado dentro do mesmo objeto. **Verdadeiro**
• Uma aplicação é considerada distribuída quando executa várias tarefas ao mesmo tempo. **Falso** - **Explicação** - Não é distribuída mas sim concorrente.
• A paralelização de um algoritmo pode ser feita através da sua decomposição funcional. **Verdadeiro**
• Um objeto do tipo SwingWorker executa as instruções do método doInBackground() de forma sequencial. **Verdadeiro**

• Em Java, uma Thread adormecida pelo método wait é acordada pelo método notify. **Verdadeiro**

Utilizando Sockets implemente um servidor que corre no endereço networkTime.edu e que forneçe o tempo local na porta 5000. O servidor tem de ser obrigatoriamente não bloqueante.
• new Date().getTime();// nº de milissegundos do tempo local do computador.

Implemente um cliente que imprima na consola o tempo fornecido pelo servidor.

Resposta:

```
//Cliente
PSVM {
	Socket socket = new Socket("network.edu", 5000);
	Scanner in = new Scanner(socket.getInputStream());
	System.out.println(in.nextLine());
	socket.close()
}

//Servidor
PSVM {
	ServerSocket server = new ServerSocket(5000);
	while(true) {
		socket s = server.accept() ;
		new thread (() -> {
		PrintStream out = new PrintStream (s.get OutputStream());
		out.println (new Date.getTime() + " ");
		s.close();
})start();
}
```

Pretende-se desenvolver um Servidor para

disponibilizar as taxas de juro Euribor a 1 semana, 1 mês, 3 meses, 6 meses e 12 meses. Estas taxas são consultadas por Clientes e atualizadas pelo Banco central. Utilizando uma tecnologia de sistemas distribuídos à sua escolha implemente:

a) [3 valores] O Servidor que está no endereço Euribor.eu e disponibiliza o porto 5.000 para atender :

Os clientes que solicitam o valor de uma taxa de juro.

O banco central que faz a atualização de uma taxa de juro

Resposta:

```
Public IEuribor extends remote {
        public double getTaxa (string tx) throws remote Exception;
        public void setTaxa (string tx, double V) throws remote Exception;
}

Public class Remote Euribor extends unicastremoteobject implements IEuribor {
    concurrent HashMap <string, double> tx = new concurrent HashMap);
    public RemoteEuribor (int p) {
    super (P))
}
  public void setTaxa (String tx, double V) throws Remote exception {
  tx.put (tx, V);
}
  public double getTaxa (String tax) throws Remote exception {
    return tx.get(tax);
}

PSMV {
    IEuribor remote = (IEuribor) Naming.lookup("x");
    Sout(remote.getMessage());
```

b) [1.0 valores] O Banco central que:

Solicita ao utilizador qual a taxa de juro que deseja alterar e o

respetivo valor
Envia as alterações para o servidor

Resposta:

```
PSVM {
IEuribor remote = (IEuribor) Naming.lookup("");
Scanner k = new Scanner(System.in);
Sout("Taxa de juro a consultar:");
String tx = k.nextLine();
Sout("Taxa");
double V = k.nextDouble();
remote.setTaxa(Tx, V);
}
```

c) [1.0 valores] O cliente que:
Solicita ao utilizador qual a taxa de juro que deseja consultar
Solicita ao servidor a referida taxa
Imprime a resposta na consola.

Resposta:

```
PSVM {
        IEuribor remote = (IEuribor) Naming.lookup("");
        Scanner k = new Scanner (System.in);
        Sout("Taxa de juro a consultar: ");
        String tx = k.nextLine();
        Sout("A taxa é de: ");
        Sout (remote.getTaxa(Tx));
}
```

Construa um programa emissor que transmita para a rede o seu endereço IP a cada minuto
utilizando o porto 5.000. O programa deve calcular o endereço de Broadcast para onde deve
enviar o seu endereço (xxx.xxx.xxx.255)

Exame Modelo