Trabalho de Programação Paralela e Distribuída em Java

Objetivo

Implementar um sistema distribuído de contagem em Java, no qual um programa D (Distribuidor) gera um grande vetor de números inteiros aleatórios do tipo byte entre -100 e 100, escolhe aleatoriamente uma posição dele, de onde toma o número, cujo número de ocorrências deve ser descoberto e o envia, juntamente com partes do vetor, a diferentes programas R (Receptores), que executam a contagem em paralelo.

A comunicação entre D e R deve ocorrer via TCP/IP, utilizando serialização de objetos e conexões persistentes. Cada servidor R deve manter sua conexão aberta até receber explicitamente um comunicado de encerramento.

1. Preparação inicial

1. Escolha 3 ou 4 computadores diferentes.
2. Descubra o endereço IP de cada computador executando no terminal:
   - Windows: ipconfig
   - Linux/macOS: ifconfig
3. Anote os IPs obtidos — eles serão utilizados pelo programa D.

2. Estrutura do sistema

O sistema é composto por dois programas distintos:
• D (Distribuidor): atua como cliente e coordena a contagem.
• R (Receptor): atua como servidor, recebendo pedidos, realizando contagens e respondendo.

Um detalhe: várias threads devem ser postas em execução para realizar simultaneamente partes da contagem designada para o R (receptor); é claro que para ter simultaneidade real é preciso por em execução uma quantidade de threads no máximo igual à quantidade de processadores que há na máquina.

3. Classes transmitidas/recebidas

As classes devem implementar o seguinte modelo de comunicação serializável:

Classe Comunicado

• Deve implementar Serializable.
• Não possui atributos nem métodos.
• Serve apenas como superclasse para Pedido, Resposta e ComunicadoEncerramento.

Classe Pedido

• Extende Comunicado.
• Atributos: byte[ ] numeros, byte procurado.

• Construtor inicializa os atributos com os valores passados.
• Método int contar( ): percorre o vetor e retorna quantas vezes o número procurado foi encontrado no vetor numeros; a contagem não será byte, será int.

Classe Resposta

• Extende Comunicado.
• Atributo: int contagem.
• Construtor recebe um int.
• Getter int getContagem( ) retorna o valor.
• Aqui, nesta classe, trabalharemos mesmo com int.

Classe ComunicadoEncerramento

• Extende Comunicado.
• Não contém atributos.
• Serve como sinal de término de comunicação entre D e R. D envia este comunicado para os R, indicando não querer processar mais nenhum vetor, o que faz com que os R fechem o transmissor, o receptor e a conexão, voltando a aceitar novas conexões.

4. Programa R (Receptor)

1. Crie um ServerSocket em uma porta fixa (ex.: 12345).
2. Aceite uma conexao.
3. Associe ObjectInputStream receptor e ObjectOutputStream transmissor à conexão.
4. Em loop, leia objetos do cliente:
   - Se for Pedido: execute contar( ), envie Resposta e continue.
   - Se for ComunicadoEncerramento: encerre a conexão e volte a aceitar conexões.

5. Programa D (Distribuidor)

1. Declare e inicialize, de forma hard coded, um vetor com os IPs dos servidores R.
2. Gere um grande vetor de inteiros e escolha aleatoriamente um número aleatório a contar.
3. Divida o vetor em partes de tamanhos semelhantes.
4. Crie uma thread para cada servidor, que:
   - Estabelece a conexão e a mantém aberta;
   - Envia sucessivos objetos Pedido, um para cada R.
   - Recebe sucessivas Respostas, uma de cada R.
5. Após o término de todas as threads, componha e exiba a resposta final.
6. Não desejando uma nova rodada de contagem em outro vetor, envia um ComunicadoEncerramento e feche todos transmissores, receptores e conexões.

6. Boas práticas exigidas e outras obrigações

• Capture e trate exceções adequadamente.
• Use Thread.join() para sincronizar as threads no cliente.
• Ofereça ao usuário a possibilidade de indicar quantos elementos quer ter no vetor grande, limitado, naturalmente, ao que seu computador suporta.
• Ofereça ao usuário a possibilidade de escrever o vetor grande na tela, para poder conferir

se seu sistema está funcionando.
• Ofereça ao usuário a possibilidade de contar também quantas vezes tem no vetor um número que não existe no vetor; neste caso, encaminhe para ser contado o número 111 e aguarde que a contagem dê 0.
• Tenha testes com vetores pequenos para poder demonstrar que seu sistema funciona
• Insira mensagens de log informativos em ambos os programas.
• Faça também um programa que realize a contagem sem paralelismo ou distribuição e meça os tempos de execução de ambos os programas para fins de comparação.
• Elabore um diário sobre o desenvolvimento da atividade, relatando a cronologia do desenvolvimento da atividade, com especial ênfase nas atividades desenvolvidas por cada membro do time de devs, incluindo timestamps. Inclua também as impresssões dos devs, bem como uma conclusão.
• Além de entregar a atividade no Canvas, demonstre-a ao professor na aula seguinte a esta de quinta-feira, dia 23/outubro.

7. Sugestões de teste

• É possível testar localmente executando várias instâncias de R em portas diferentes.
• Teste posteriormente em máquinas distintas, na mesma rede local.
• Exemplo de logs:
[R] Pedido recebido do cliente 172.16.21.50
[D] Enviando Pedido para 172.16.21.22...
[D] Resposta recebida: número não encontrado.

8. Entrega

• Código-fonte completo dos programas D e R.
• Classes Comunicado, Pedido, Resposta e ComunicadoEncerramento.
• Capturas de tela mostrando os programas em execução.
• Relato breve (até 10 linhas) sobre os testes realizados.

9. Como descobrir quantos processadores há na máquina?

```
int quantidade = Runtime.getRuntime().availableProcessors ( );
```

10. Como sortear um número aleatório entre min e max (incluindo min e max)

```
int aleatorio = ((int)(Math.random( )*(max-min)))+min;
```

11. Como descobrir o que é um vetor grande e como medir o tempo de execução dum programa?

```
public class MaiorVetorAproximado {
   public static void main(String[ ] args) {
      System.out.println("Estimando o maior tamanho possível de vetor em Java...");
      long inicio = System.currentTimeMillis();
```

```
        int tamanho = 1_000_000;  // começa com 1 milhão
        int ultimoBemSucedido = 0;

        while (true) {
          try {
            byte[ ] vetor = new byte[tamanho];
            ultimoBemSucedido = tamanho;
            vetor = null;  // libera
            System.gc();

            // aumenta o tamanho em 50% para a próxima tentativa
            if (tamanho > Integer.MAX_VALUE / 3 * 2) break;

            tamanho /= 2;
            tamanho *= 3;

            System.out.printf("Alocado com sucesso: %,d elementos%n", ultimoBemSucedido);
          } catch (OutOfMemoryError e) {

           System.out.printf("Falhou em %,d elementos%n", tamanho);
           break;
          }
        }

        long fim = System.currentTimeMillis();
        System.out.println("\nMaior vetor que coube (aproximadamente): "+
                                  String.format("%,d", ultimoBemSucedido));
        System.out.printf("Memória estimada: %.2f MB%n",
                                 ultimoBemSucedido * 1.0 / (1024 * 1024));
        System.out.printf("Tempo total: %.2f segundos%n", (fim - inicio) / 1000.0);
      }
    }

    // rode o programa com o comando:
    // java -Xmx8G MaiorVetorAproximado
    // para disponibilizar 8gb de memória para uso do Java, para criar o vetor grande
    // não ponha 8gb, se 8gb for toda memória que seu computador tem
    // subtraia ao menos 3gb da memória que seu computador tem
```