Trabalho I de Fundamento de Processamento Paralelo e Distribuído

Grupo: Dhruv Babani,Eduardo Barcellos,Eduardo Bregalda

Primeiro código(Algoritmo de Dijkstra):

Este código implementa uma solução para o problema clássico do jantar dos filósofos usando o algoritmo de Dijkstra. Cada filósofo é modelado como uma thread separada. Cada filósofo possui um identificador único (id) e é associado a dois semáforos: garfo_esquerdo e garfo_direito.

O funcionamento geral da solução segue um ciclo de vida para cada filósofo, que consiste em três etapas principais: pensar, pegar garfos e comer. Inicialmente, o filósofo está pensando e aguarda um período de tempo aleatório simulando esse estado. Quando decide comer, ele tenta adquirir os garfos esquerdo e direito, garantindo que ambos estejam disponíveis.

Isso é feito usando os semáforos correspondentes. Se o filósofo não conseguir adquirir ambos os garfos, ele aguardará até que ambos estejam disponíveis. Para pegar os garfos, os métodos pegarGarfo_esquerdo() e pegarGarfo_direito() são usados, onde cada um deles verifica a disponibilidade do garfo correspondente por meio de semáforos e, em seguida, adquire o garfo. Após adquirir ambos os garfos, o filósofo entra no estado de comer, simulando esse estado aguardando um período de tempo aleatório. Em seguida, ele libera os garfos usando o método devolver_garfo() para permitir que outros filósofos possam comê-los.

Essa solução evita deadlock, pois um filósofo só pegará garfos se ambos estiverem disponíveis e libera os garfos após comer. Isso garante que outros filósofos possam pegá-los. A solução também evita starvation, pois não há garantia de que um filósofo específico sempre será o próximo a comer.

Segundo código(Algoritmo de Chandy Misra):

O segundo código implementa uma solução alternativa para o problema do jantar dos filósofos, usando um sistema de fila de prioridade. Nesta solução, os filósofos entram em um estado chamado "COM_FOME" antes de tentar pegar os garfos. A transição para "COM_FOME" ocorre de maneira ordenada por meio de um loop que verifica o estado de um filósofo e o altera se estiver "PENSANDO".

Antes de tentar pegar os garfos, os filósofos verificam se ambos estão disponíveis simultaneamente. Se estiverem, o filósofo pode pegá-los ao mesmo tempo, mudando do estado "COM_FOME" para "COMENDO". Após terminar de comer, o filósofo retorna ao estado "PENSANDO" e libera os garfos para permitir que outros filósofos comam.

Esta implementação também evita deadlock, pois os filósofos aguardam a disponibilidade de ambos os garfos antes de tentar pegá-los. Evita starvation, pois todos os filósofos têm a oportunidade de entrar no estado "COM_FOME" e tentar pegar os garfos.

Trabalho I de Fundamento de Processamento Paralelo e Distribuído

Grupo: Dhruv Babani,Eduardo Barcellos,Eduardo Bregalda

Resultados do Primeiro código:

```
Filosofo 4 está PENSANDO.
Filosofo 3 está SEGURANDO o garfo direito
Filosofo 3 está COMENDO.
Filosofo 0 está SEGURANDO o garfo esquerdo
Filosofo 0 está ESPERANDO o garfo direito
Filosofo 4 está ESPERANDO o garfo esquerdo
Filosofo 3 soltou os garfos.
Filosofo 3 está PENSANDO.
Filosofo 2 está SEGURANDO o garfo direito
Filosofo 2 está COMENDO.
Filosofo 4 está SEGURANDO o garfo esquerdo
Filosofo 4 está ESPERANDO o garfo direito
Filosofo 3 está ESPERANDO o garfo esquerdo
Filosofo 2 soltou os garfos.
Filosofo 2 está PENSANDO.
```

```
Filosofo 2 está PENSANDO.
Filosofo 1 está SEGURANDO o garfo direito
Filosofo 1 está COMENDO.
Filosofo 3 está SEGURANDO o garfo esquerdo
Filosofo 3 está ESPERANDO o garfo direito
Filosofo 2 está ESPERANDO o garfo esquerdo
Filosofo 1 soltou os garfos.
Filosofo 1 está PENSANDO.
Filosofo 1 está SEGURANDO o garfo esquerdo
Filosofo 1 está ESPERANDO o garfo direito
Filosofo 2 está SEGURANDO o garfo esquerdo
Filosofo 2 está ESPERANDO o garfo direito
```

Resultados do Segundo Código

```
 Tempo atual : 99

 Tempo médio que o  Filósofo 1  demorou para comer ( ficou com fome ) : 5.2

 Tempo médio que o  Filósofo 2  demorou para comer ( ficou com fome ) : 4.3333333333
33333

 Tempo médio que o  Filósofo 3  demorou para comer ( ficou com fome ) : 3.7142857142
857144

 Tempo médio que o  Filósofo 4  demorou para comer ( ficou com fome ) : 3.25

 Tempo médio que o  Filósofo 5  demorou para comer ( ficou com fome ) : 3.7142857142
857144

 Tempo médio para os filósofos comerem 2.9696969696969697
```

Trabalho I de Fundamento de Processamento Paralelo e Distribuído

Grupo: Dhruv Babani,Eduardo Barcellos,Eduardo Bregalda

Primeiro Código:

```
import java.util.Random;
import java.util.concurrent.Semaphore;

class Filosofos implements Runnable {
  // Usados para varirar quanto tempo um filosofo pensa antes de comer
  // e quanto tempo ele come
  private Random number = new Random();

  // Utilitarios dos filosofos
  private final int id;
  private final Semaphore garfo_esquerdo;
  private final Semaphore garfo_direito;

  public Filosofos(int id, Semaphore garfo_esquerdo, Semaphore garfo_direito) {
    this.id = id;
    this.garfo_esquerdo = garfo_esquerdo;
    this.garfo_direito = garfo_direito;
  }

  // Ciclo infinito de cada filosofo executando em threads separadas
  @Override
  public void run() {
    // Clausula de erro obrigatória
    try {
      while (true) {
        pensar();
        pegarGarfo_esquerdo();
        pegarGarfo_direito();
        comer();
        devolver_garfo();
      }
    } catch (InterruptedException e) {
      System.out.println("Filosofo " + " foi interropido.\n");
    }
  }

  // Modelo de pensamento. Define um tempo aleatório pro filosofo fazer isso
  private void pensar() throws InterruptedException {
    System.out.println("Filosofo " + id + " está PENSANDO.\n");
    Thread.sleep(number.nextInt(10)); // Simula o tempo de execução
  }

  // Analisa a quantidade de permissões para poder pegar o garfo
  private void pegarGarfo_esquerdo() throws InterruptedException {
    if (garfo_esquerdo.availablePermits() == 0) {
      System.out.println("Filosofo " + id + " está ESPERANDO o garfo esquerdo\n");
    }
    garfo_esquerdo.acquire();
```

Trabalho I de Fundamento de Processamento Paralelo e Distribuído

Grupo: Dhruv Babani,Eduardo Barcellos,Eduardo Bregalda

```
    System.out.println("Filosofo " + id + " está SEGURANDO o garfo esquerdo\n");
  }

  private void pegarGarfo_direito() throws InterruptedException {
    if (garfo_direito.availablePermits() == 0) {
      System.out.println("Filosofo " + id + " está ESPERANDO o garfo direito\n");
    }

    garfo_direito.acquire();
    System.out.println("Filosofo " + id + " está SEGURANDO o garfo direito\n");
  }

  //
  private void comer() throws InterruptedException {
    System.out.println("Filosofo " + id + " está COMENDO.\n");
    Thread.sleep(number.nextInt(10));
  }

  // Libera os garfos para que os outros filosofos possam pegar
  private void devolver_garfo() {
    garfo_esquerdo.release();
    garfo_direito.release();
    System.out.println("Filosofo " + id + " soltou os garfos.\n");
  }

}
import java.util.concurrent.Semaphore;

public class JantarDosFilosofos {

   private static final int N = 5; // Número de filósofos

   public static void main(String[] args) {
      // Cada garfo será um semáforo
      Semaphore[] garfo = new Semaphore[N];

      for (int i = 0; i < N; i++) {
         garfo[i] = new Semaphore(1);
      }

      // Cria os filósofos e inicia cada um executando a sua thread
      Filosofos[] filosofo = new Filosofos[N];
      for (int i = 0; i < N; i++) {
         filosofo[i] = new Filosofos(i, garfo[i], garfo[(i + 1) % N]);
         new Thread(filosofo[i]).start();
      }

   }
}
```

Trabalho I de Fundamento de Processamento Paralelo e Distribuído

Grupo: Dhruv Babani,Eduardo Barcellos,Eduardo Bregalda

Segundo Código:

```
import java.io.IOException;
import java.util.Random;

import application.Filosofo;
import application.AcaoFilosofo.Acao;

public class JantarDosFilosofos {

    // Declarando variaveis a serem utilizadas
    private Random random = new Random(); // Variavel aleatoria para gerar filosofo com fome
    private Filosofo[] filosofos = new Filosofo[5]; // Variavel do tipo filosofo para armazenar os 5 filosofos presentes
                                                    // com as funcoes necessarias .
    private int tempoAtual; // Variavel para demarcar o tempo atual em que se encontra a mesa
    private int x, y, z; // Variaveis genericas
    private double[] contaFomeLocal = new double[5]; // Variavel para contar quantas vezes cada filosofo ficou com fome
    private double fomeTotal = 0; // Variavel para somar o tempo com fome de todos os filosofos
    private double contaFomeTotal = 0; // Variavel para somar todas as vezes que os filosofos ficaram com fome
    private boolean[] garfo = new boolean[5]; // Variavel que indica quais garfos estao disponiveis no estado em que se
                                              // encontra a mesa
    private final int tempo = 100; // Tempo limite da execucao

    public JantarDosFilosofos() {

        System.out.println("\nJantar dos Filosofos \n");

        preparaJantar();

        for (tempoAtual = 1; tempoAtual < tempo; tempoAtual++) { // Execucao principal do programa

            filosofoTerminaRefeicao();
            filosofosTentamComer();
            if (tempoAtual % 3 == 0) {
                geraFilosofoComFome();
            }

            mostraMesa();
        }

        for (int i = 0; i < 5; i++) {
            System.out.println(
                    " \n Tempo médio que o " + filosofos[i].getNome() + " demorou para comer ( ficou com fome ) : "
                            + filosofos[1].getTempoComFome() / filosofos[i].getContaFome() + " \n ");

            fomeTotal = fomeTotal + filosofos[i].getTempoComFome();
            contaFomeTotal = contaFomeTotal + filosofos[i].getContaFome();
        }
        System.out.println(" \n Tempo médio para os filósofos comerem " + fomeTotal / contaFomeTotal + " \n ");
```

Trabalho I de Fundamento de Processamento Paralelo e Distribuído

Grupo: Dhruv Babani,Eduardo Barcellos,Eduardo Bregalda

```
}

public int getTempoAtual() {
    return tempoAtual;
}

public void preparaJantar() { // Instancia os filosofos e " coloca os garfos na mesa "
    for (x = 0; x < 5; x++) {
        filosofos[x] = new Filosofo(Acao.PENSANDO);
        garfo[x] = true;
        switch (x) {
            case 0:
                filosofos[0].setNome(" Filósofo 1 ");
                contaFomeLocal[0] = 0;
                break;
            case 1:
                filosofos[1].setNome(" Filósofo 2 ");
                contaFomeLocal[1] = 0;
                break;
            case 2:
                filosofos[2].setNome(" Filósofo 3 ");
                contaFomeLocal[2] = 0;
                break;
            case 3:
                filosofos[3].setNome(" Filósofo 4 ");
                contaFomeLocal[3] = 0;
                break;
            case 4:
                filosofos[4].setNome(" Filósofo 5 ");
                contaFomeLocal[4] = 0;
                break;
        }
    }
}

public void filosofoTerminaRefeicao() {
    for (y = 0; y < 5; y++) {
        if (filosofos[y].getAcao() == Acao.COMENDO) { // Condicao para mostrar qual ( is ) filosofos esta ( ao )
                                                      // comendo
            filosofos[y].addTempoComendo(); // Aumenta o tempo que o filosofo passa comendo
            if (filosofos[y].getTempoComendo() == 5) {
                synchronized (filosofos[y]) {
                    if (y == 4) {
                        garfo[0] = true;
                    } else {
                        garfo[y + 1] = true;
                    }
                    garfo[y] = true;
                    filosofos[y].setTempoComendo(0);
                    filosofos[y].setAcao(Acao.PENSANDO);
```

Trabalho I de Fundamento de Processamento Paralelo e Distribuído

Grupo: Dhruv Babani,Eduardo Barcellos,Eduardo Bregalda

```
                    try {
                        Thread.sleep(50);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    filosofos[y].notifyAll();
                    filosofos[y].setSegurandoGarfo(false);
                }
            }
        }
    }

    public void geraFilosofoComFome() { // Funcao para mudar acao de um filosofo de PENSANDO para COM FOME
        y = 0;
        for (x = 0; x < 5; x++) {
            if (filosofos[x].getAcao() != Acao.PENSANDO) {
                y++;
            }
        }
        if (y != 5) {

            z = random.nextInt(5);
            while ((filosofos[z].getAcao() != Acao.PENSANDO)) {
                z = random.nextInt(5);
            }
            filosofos[z].setAcao(Acao.COM_FOME);
            contaFomeLocal[z]++;
            filosofos[z].setContaFome(contaFomeLocal[z]);
        }
    }

    public void filosofosTentamComer() {
        for (y = 0; y < 5; y++) {

            if (y == 4) {
                x = 0;
            } else {
                x = y + 1;
            }
            if (filosofos[y].getAcao() == Acao.COM_FOME) {
                if (filosofos[y].pegaGarfos(garfo[y], garfo[x])) { // Garante que os dois garfos ao lado do filosofo COM
                                                                   // FOME estão disponiveis , caso não estejam gera
                                                                   // sleep em pegaGartos
                    garfo[x] = false;
                    garfo[y] = false;
                }
            }
        }
    }
```

Trabalho I de Fundamento de Processamento Paralelo e Distribuído

Grupo: Dhruv Babani,Eduardo Barcellos,Eduardo Bregalda

```java
    public String temGarfo(boolean garfoString) {
        String trocaString;

        if (garfoString == false) {
            trocaString = " Sem garfo ";
        } else {
            trocaString = " Com garfo ";
        }
        return trocaString;
    }

    public void mostraMesa() {// Menu principal que mostra o funcionamento da mesa dos filosofos
        System.out.println("\n-------------------------------------------------------------\n");

        System.out.println(" \n " + filosofos[0].getNome() + " " + filosofos[0].getAcao());
        System.out.println(" \n 1 : " + temGarfo(garfo[0]) + " 2: " + temGarfo(garfo[1]));
        System.out.println(" \n " + filosofos[4].getNome() + " " + filosofos[4].getAcao() + " " + filosofos[1].getNome()
                + "" + filosofos[1].getAcao());
        System.out.println(" \n 5 : " + temGarfo(garfo[4]) + " 3: " + temGarfo(garfo[2]));
        System.out.println("  \n  " + filosofos[3].getNome() + " " + filosofos[3].getAcao() + " "
                + filosofos[2].getNome() + " " + filosofos[2].getAcao());
        System.out.println(" \n 4: " + temGarfo(garfo[3]));
        System.out.println(" \n \n Tempo atual : " + tempoAtual);

    }

    public static void main(String[] args) {
        new JantarDosFilosofos();
    }

}
```

-=-

```java
package application;

import application.AcaoFilosofo.Acao;

public class Filosofo extends Thread {

    private Acao acao;
    private String nome;
    private int tempoComendo = 0;
    private double contaFome = 0;
    private boolean segurandoGarfo = false;

    public int tempoComFome;
```

Trabalho I de Fundamento de Processamento Paralelo e Distribuído

Grupo: Dhruv Babani,Eduardo Barcellos,Eduardo Bregalda

```
public Filosofo(Acao s) {
    acao = s;
}

public Acao getAcao() {
    return acao;
}

public void setAcao(Acao acao) {
    this.acao = acao;
}

public void setNome(String nome) {
    this.nome = nome;
}

public String getNome() {
    return nome;
}

public int getTempoComendo() {
    return tempoComendo;
}

public void setTempoComendo(int n) {
    tempoComendo = n;
}

public void addTempoComendo() {
    tempoComendo++;
}

public void setSegurandoGarfo(boolean b) {
    segurandoGarfo = b;
}

public boolean getSegurandoGarfo() {
    return segurandoGarfo;
}

public boolean pegaGarfos(boolean garfoD, boolean garfoE) {
    synchronized (this) {

        if ((garfoD == true) && (garfoE == true)) {
            tempoComFome++;
            acao = Acao.COMENDO;

            return true;

        } else {
```

Trabalho I de Fundamento de Processamento Paralelo e Distribuído

Grupo: Dhruv Babani,Eduardo Barcellos,Eduardo Bregalda

```
            try {
                tempoComFome++;
                sleep(10);
            } catch (InterruptedException e) {
                System.out.println(" Erro : " + e);
            }
            return false;
        }
    }
  }

  public double getTempoComFome() {
    return tempoComFome;

  }

  public double getContaFome() {
    return contaFome;
  }

  public void setContaFome(double i) {
    contaFome = i;
  }
}
```