**1. Quản lý vùng găng**

Chương trình xem mỗi triết gia là 1 tiến trình, chopstick là tài nguyên chung cần được bảo vệ.

Biến leftFork và rightFork được xây dựng trong class Philosopher để quản lý tài nguyên dùng chung là các forks (được khai báo ở class DiningProblem), mỗi fork được tạo ra sẽ có 1 phương thức khóa availablePermits() để đánh dấu :  
• leftFork.availablePermits(), rightFork.availablePermits() = 1 => dĩa bên trái (phải) chưa được sử dụng.  
• leftFork.availablePermits(), rightFork.availablePermits() = 0 => dĩa bên trái (phải) đang được sử dụng.

**2. Giải pháp xử lý deadlock**

Vấn đề deadlock có thể được tránh khỏi bằng viêc xây dụng phương thức wait() và signal() sao cho vòng tròn đợi không xảy ra.

Semaphores là các biến số nguyên được sử dụng để giải quyết vấn đề phần quan trọng bằng cách sử dụng hai phép toán nguyên tử, chờ và tín hiệu được sử dụng để đồng bộ hóa quy trình. Các định nghĩa của chờ và tín hiệu như sau:

1. Wait - Thao tác chờ làm giảm giá trị của đối số S, nếu nó là số dương. Nếu S âm hoặc bằng 0, thì không có thao tác nào được thực hiện.

|  |
| --- |
| sem\_wait(Semaphore s){  while(s <= 0);  s--;  } |

1. Signal - Phép toán tín hiệu tăng giá trị của đối số S.

|  |
| --- |
| sem\_signal(Semaphore s){  s++;  } |

Khi một tiến trình đang chạy phần quan trọng thì một quá trình khác không thể truy cập vào phần quan trọng cho đến khi sem\_signal chạy và làm cho biến s bằng 1.

Phương thức Wait() và Signal() được xây dựng để quản lí tiến trình ra vào vùng găng. Với cách xây dựng như trên thì 2 tiến trình cạnh nhau có số thứ tự chẳn lẽ khác nhau sẽ có thứ tự lấy các chopstick theo thứ tự khác nhau.

**3. Chương trình**

Chương trình xậy dựng gồm 2 class:

- Class Philosopher: class tạo các luồng và các hoạt động của luồng.

- Class DiningProblem: Class chứa hàm main và tài nguyên dùng chung (fork).

**Nhắc lại bài toán về bữa tối nhà triết học**

Bài toán được phát biểu như sau: "Khi một triết gia suy nghĩ, ông ta không giao tiếp với các triết gia khác. Thỉnh thoảng, một triết gia cảm thấy đói và cố gắng chọn hai chiếc dĩa gần nhất (hai chiếc dĩa nằm giữa ông ta với hai láng giềng trái và phải). Một triết gia có thể lấy chỉ một chiếc dĩa tại một thời điểm. Chú ý, ông ta không thể lấy chiếc dĩa mà nó đang được dùng bởi người láng giềng. Khi một triết gia đói và có hai chiếc dĩa cùng một lúc, ông ta ăn mà không đặt dĩa xuống. Khi triết gia ăn xong, ông ta đặt dĩa xuống và bắt đầu suy nghĩ tiếp."

**Thuật toán:**

|  |
| --- |
| repeat  wait(fork[i]);  wait(fork[(i+1) mod 5]);  . . .  eat  . . .  signal(fork[i]);  signal(fork[(i+1) mod 5]);  . . .  think  . . .  until false; |

**Mã nguồn:**

Tạo một lớp Thread có tên là Philosopher:

|  |
| --- |
| class Philosopher implements Runnable {  private int id;  private int eattime;  private int amount=0;  private Semaphore leftFork;  private Semaphore rightFork;  //Tạo constructor cho lớp Philosopher  public Philosopher(int id, Semaphore leftFork, Semaphore rightFork) {  this.id = id;  this.leftFork = leftFork;  this.rightFork = rightFork;  }  public void run() {  ...  }  //các code tiếp theo  } |

Các triết gia sẽ nghĩ với thời gian ngẫu nhiên thông qua method think:

|  |
| --- |
| private void think() throws InterruptedException {  System.out.println("Philosopher " + (id + 1) + " is thinking.");  System.out.flush();  Thread.sleep(new Random().nextInt(10));  } |

Sau đó tạo phương thức pickUpLeftFork và pickUpRightFork. Giá trị của các semaphore ban đầu là 1. Khi semaphore = 1 thì triết gia có thể lấy dĩa. Sau đó sử dụng phương thức aqquire() trong lớp semaphore để khóa dĩa .

|  |
| --- |
| private void pickUpLeftFork() throws InterruptedException{  if(leftFork.availablePermits() == 0) {  System.out.println("Philosopher " + (id + 1) + " is waiting for left fork");  }  leftFork.acquire();  System.out.println("Philosopher " + (id + 1) + " is holding left fork.");  }  private void pickUpRightFork() throws InterruptedException{  if(rightFork.availablePermits() ==0){  System.out.println("Philosopher " + (id + 1) + " is waiting for right fork");  }  rightFork.acquire();  System.out.println("Philosopher " + (id + 1) + " is holding right fork.");  } |

Khóa chỉ có thể được gỡ khi sử dụng phương thức release trong lớp semaphore. Nếu biến semaphore = 0 thì các triết gia không được lấy dĩa mà phải đợi.

|  |
| --- |
| private void putDownFork() {  leftFork.release();  rightFork.release();  System.out.println("Philosopher " + (id + 1) + " ate "+amount+"% and"+" released left and right forks");  } |

Tạo một phương thức để ăn có tên eat

|  |
| --- |
| private void eat() throws InterruptedException {  System.out.println("Philosopher " + (id + 1) + " is eating.");  System.out.flush();  do{  eattime = new Random().nextInt(10);  //generate a random value for eat time which is greater than 0  } while (eattime <= 0);  if (amount + eattime \* 5 > 100) {  eattime = (100 - amount) / 5;  amount = 100;  } else {  amount = amount + eattime \* 5;  }  Thread.sleep(eattime);  } |

Giả sử mỗi triết gia ăn 20% khẩu phần ăn của mình trong mỗi lần ăn.

|  |
| --- |
| public class DiningProblem {  private static final int n = 5;    public static void main(String[] args) {  Semaphore[] fork = new Semaphore[n];  for (int i = 0; i < n; i++) {  fork[i] = new Semaphore(1);  }  Philosopher[] philosophers = new Philosopher[n];  for (int i = 0; i < n; i++) {  philosophers[i] = new Philosopher(i, fork[i], fork[(i + 1) % n]);  new Thread(philosophers[i]).start();  }  }  } |

n là số triết học. Tạo một mảng biến semaphore cho mỗi dĩa, và cho chúng ở trạng thái đang chờ được sử dụng (biến semaphore = 1). Sau đó tạo một mảng các triết gia và khởi tạo với các tham số lần lượt là id, dĩa trái và dĩa phải.

**4. Kết quả chương trình**

|  |
| --- |
| Philosopher 1 is thinking.  Philosopher 1 is holding left fork.  Philosopher 1 is holding right fork.  Philosopher 1 is eating.  Philosopher 1 ate 40% and released left and right forks  Philosopher 1 is thinking.  Philosopher 2 is thinking.  Philosopher 1 is holding left fork.  Philosopher 1 is holding right fork.  Philosopher 1 is eating.  Philosopher 1 ate 45% and released left and right forks  Philosopher 1 is thinking.  Philosopher 1 is holding left fork.  Philosopher 1 is holding right fork.  Philosopher 1 is eating.  Philosopher 3 is thinking.  Philosopher 2 is waiting for left fork  Philosopher 3 is holding left fork.  Philosopher 3 is holding right fork.  Philosopher 3 is eating.  Philosopher 2 is holding left fork.  Philosopher 1 ate 85% and released left and right forks  Philosopher 1 is thinking.  Philosopher 2 is waiting for right fork  Philosopher 4 is thinking.  Philosopher 1 is holding left fork.  Philosopher 2 is holding right fork.  Philosopher 3 ate 30% and released left and right forks  Philosopher 2 is eating.  Philosopher 1 is waiting for right fork... |

Nguồn tham khảo: https://github.com/EANimesha/Dinning-Philosopher-Problem