PROJEKT

Jacek Maruszak

Zadanie - Zakład produkcyjny

Treść zadania:

Założenia:

Zakład produkcyjny produkuje na kilku liniach wyroby tego samego typu korzystając z dwóch rodzajów surowców. Surowce różnego rodzaju przechowywane są w oddzielnych magazynach, które zaopatrywane są przez jeden samochód w sposób wyłączny. Surowce danego rodzaju dowożone są po spadku zapasu w magazynie poniżej ustalonego poziomu. Produkcja wyrobów na linii zajmuje losowy czas. Wyroby gotowe trafiają z linii do magazynu wyjściowego, z którego co losowy czas są zabierane w formie paczek (ustalona liczba sztuk).

Kod źródłowy:

```
import java.util.Random;
import java.util.concurrent.Semaphore;
class ProductionLine {
    private Semaphore semaphore1;
    private Semaphore semaphore2;
    private int resource1;
    private int resource2;
    private int finishedProducts;
    private int resource1Threshold;
    private int resource2Threshold;
    private Random rand;

    public ProductionLine(int resource1Threshold, int
resource2Threshold) {
        this.semaphore1 = new Semaphore(resource1Threshold);
        this.semaphore2 = new Semaphore(resource2Threshold);
        this.resource1 = 0;
        this.resource2 = 0;
        this.finishedProducts = 0;
        this.resource1Threshold = resource1Threshold;
        this.resource2Threshold = resource2Threshold;
        this.rand = new Random();
    }

    public synchronized void produce() throws InterruptedException {
        if (resource1 < resource1Threshold || resource2 <
resource2Threshold) {
            System.out.println("Brak surowców. POTRZEBNA DOSTAWA");
            return;
        }
```

```
        while(resource1 >= resource1Threshold) {
            int productionTime = rand.nextInt(1000);
            resource1 -= resource1Threshold;
            try {
                semaphore1.acquire();
                System.out.println("Wytworzono [Produkt 1]" + ",
Zostało: " + resource1);
                Thread.sleep(productionTime);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            finishedProducts += 1;
        }
        while(resource2 >= resource2Threshold) {
            int productionTime2 = rand.nextInt(1000);
            resource2 -= resource2Threshold;
            try {
                semaphore2.acquire();
                System.out.println("Wytworzono [Produkt 2]" + ",
Zostało: " + resource2);
                Thread.sleep(productionTime2);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            finishedProducts += 1;
        }

        System.out.println("----------(END OF PRODUKCJA NA LINII)-----
-----");
        Thread.sleep(2000);
    }

    public synchronized void addResource1(int amount) {
        resource1 += amount;
        semaphore1.release(amount);
        System.out.println("Dostarczono Surowiec 1" + ", W ilości: " +
amount);
        notifyAll();
    }

    public synchronized void addResource2(int amount) {
        resource2 += amount;
```

```
        semaphore2.release(amount);
        System.out.println("Dostarczono Surowiec 2" + ", W ilości: " +
amount);
        notifyAll();
    }
    public int getResource1() {
        return resource1;
    }

    public int getResource2() {
        return resource2;
    }

    public int getFinishedProducts() {
        return finishedProducts;
    }
    public synchronized int takeFinishedProducts(int amount) {
        while (finishedProducts < amount) {
            try {
                wait();

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        finishedProducts -= amount;
        notifyAll();
        System.out.println("Brak materialow. POTRZEBNA DOSTAWA!");

        return amount;
    }
}

public class ProductionSimulation {
    public static void main(String[] args) throws InterruptedException
{
        System.out.println("[PRODUKT 1] - WYMAGA 10 ZASOBÓW [S1]");
        System.out.println("[PRODUKT 2] - WYMAGA 20 ZASOBÓW [S2]\n");

        Thread.sleep(3000);

        ProductionLine line = new ProductionLine(10, 20);
        ProductionLine line2 = new ProductionLine(10, 20);
```

```
        SupplyTruck truck1 = new SupplyTruck(line);
        ShippingTruck truck2 = new ShippingTruck(line);
        Thread supplyThread = new Thread(() -> truck1.run());
        Thread shippingThread = new Thread(() -> truck2.run());
        supplyThread.start();
        shippingThread.start();

        while (true) {
            line.produce();
            line2.produce();
        }
    }
}
```

```
import java.util.Random;
class ShippingTruck {
    private ProductionLine line;
    private Random rand;

    public ShippingTruck(ProductionLine line) {
        this.line = line;
        this.rand = new Random();
    }

    public void run() {
        while (true) {
            int packageAmount = rand.nextInt(100);
            line.takeFinishedProducts(packageAmount);
            int shippingTime = rand.nextInt(100);
            try {
                Thread.sleep(shippingTime);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
import java.util.Random;
class SupplyTruck {
    private ProductionLine line;
    private Random rand;
    public SupplyTruck(ProductionLine line) {
```

```
        this.line = line;
        this.rand = new Random();
    }
    public void run() {
        while (true) {
            int resource1Amount = rand.nextInt(100);
            int resource2Amount = rand.nextInt(100);
            line.addResource1(resource1Amount);
            line.addResource2(resource2Amount);
            int supplyTime = rand.nextInt(100);
            try {
                Thread.sleep(supplyTime);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Założenia:

W rozwiązaniu przyjęto, że linia produkcyjna wytwarza produkt z konkretnej liczby danego surowca z tym, że liczba ta może być zmieniona. Linie równolegle produkują inne produkty, ich załadowanie do ciężarówki transportowej (shippingtruck) następuje w tym samym momencie. Analogicznie dla ciężarówki dowożącej surowce (supplytruck), różne surowce dowożone są w tym samym momencie, w tym samym "kursie" ciężarówki. Produkt wytwarzany jest tylko w momencie, gdy w magazynie danego surowca jest jego wystarczająca liczba. Ponadto w sytuacji, gdy brakuje surowca, natychmiastowo wysyłany jest komunikat o niemożności wytworzenia produktu i nakazie dostawy jego dodatkowej ilości. Naturalnie brak surowca nie powoduje braku możliwości produkcji produktu, który go nie potrzebuje. Produktów produkowana jest maksymalna liczba, na którą pozwalają surowce. Pozostałe surowce, nieużyte na danym etapie produkcyjnym, zostają na stanie, czekając na zwiększenie ich ilości

kolejną dostawą. Ilość danego surowca w dostawie jest naturalnie losowa, inna co dostawę.

Wykaz współdzielonych zasobów:

- Obiekty ‘ProductionLine’ pełniące również funkcję magazynu dla surowca, posiadające minimalny próg surowców dla produkcji danego produktu oraz ich aktualny stan (jak i ilość produktów wytworzonych na linii);

Wykaz wyróżnionych punktów synchronizacji:

```
while(resource1 >= resource1Threshold) {
    int productionTime = rand.nextInt(1000);
    resource1 -= resource1Threshold;
    try {
        semaphore1.acquire();
        System.out.println("Wytworzono [Produkt 1]" + ", Zostało: " +
resource1);
        Thread.sleep(productionTime);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    finishedProducts += 1;

}
```

- Podniesienie semafora (semaphore1) dla progu surowców potrzebnych do wytworzenia Produktu 1;

```
while(resource2 >= resource2Threshold) {
    int productionTime2 = rand.nextInt(1000);
    resource2 -= resource2Threshold;
    try {
        semaphore2.acquire();
        System.out.println("Wytworzono [Produkt 2]" + ", Zostało: " +
resource2);
        Thread.sleep(productionTime2);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    finishedProducts += 1;
}
```

- Analogicznie podniesienie semafora (semaphore2) dla progu surowców potrzebnych do wytworzenia Produktu 2;

```
public synchronized void addResource1(int amount) {
    resource1 += amount;
    semaphore1.release(amount);
    System.out.println("Dostarczono Surowiec 1" + ", W ilości: " +
amount);
    notifyAll();
}
```

- Opuszczenie semafora (semaphore1) w celu umożliwienia przybycia nowo przebytych materiałów (surowców)

```
public synchronized void addResource2(int amount) {
    resource2 += amount;
    semaphore2.release(amount);
    System.out.println("Dostarczono Surowiec 2" + ", W ilości: " +
amount);
    notifyAll();
}
```

- Analogicznie opuszczenie semafora (semaphore2) w celu umożliwienia przybycia nowo przebytych materiałów (surowców)

Wykaz obiektów synchronizacji:

- klasa ProductionLine – utworzone domyślnie obiekty linia1 oraz linia2

Wykaz procesów sekwencyjnych:

- SupplyThread
- ShippingThread
- Macierzysty, z klasą ProductionSimulation

Listing programu:

1)

```
public class ProductionSimulation {
    public static void main(String[] args) throws InterruptedException {
        System.out.println("[PRODUKT 1] - WYMAGA 10 ZASOBÓW [S1]");
        System.out.println("[PRODUKT 2] - WYMAGA 20 ZASOBÓW [S2]\n");

        Thread.sleep(3000);

        ProductionLine line = new ProductionLine(10, 20);
        ProductionLine line2 = new ProductionLine(10, 20);
        SupplyTruck truck1 = new SupplyTruck(line);
        ShippingTruck truck2 = new ShippingTruck(line);
        Thread supplyThread = new Thread(() -> truck1.run());
        Thread shippingThread = new Thread(() -> truck2.run());
        supplyThread.start();
        shippingThread.start();

        while (true) {
            line.produce();
            line2.produce();
        }
    }
}
```

2)

```
class ProductionLine {
    private Semaphore semaphore1;
    private Semaphore semaphore2;
    private int resource1;
    private int resource2;
    private int finishedProducts;
    private int resource1Threshold;
    private int resource2Threshold;
    private Random rand;

    public ProductionLine(int resource1Threshold, int resource2Threshold) {
        this.semaphore1 = new Semaphore(resource1Threshold);
        this.semaphore2 = new Semaphore(resource2Threshold);
        this.resource1 = 0;
        this.resource2 = 0;
        this.finishedProducts = 0;
        this.resource1Threshold = resource1Threshold;
```

```
        this.resource2Threshold = resource2Threshold;
        this.rand = new Random();
    }

    public synchronized void produce() throws InterruptedException {
        if (resource1 < resource1Threshold || resource2 < resource2Threshold)
{
            System.out.println("Brak surowców. POTRZEBNA DOSTAWA");
            return;
        }
        while(resource1 >= resource1Threshold) {
            int productionTime = rand.nextInt(1000);
            resource1 -= resource1Threshold;
            try {
                semaphore1.acquire();
                System.out.println("Wytworzono [Produkt 1]" + ", Zostało: " +
resource1);
                Thread.sleep(productionTime);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            finishedProducts += 1;
        }
        while(resource2 >= resource2Threshold) {
            int productionTime2 = rand.nextInt(1000);
            resource2 -= resource2Threshold;
            try {
                semaphore2.acquire();
                System.out.println("Wytworzono [Produkt 2]" + ", Zostało: " +
resource2);
                Thread.sleep(productionTime2);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            finishedProducts += 1;
        }

        System.out.println("----------(END OF PRODUKCJA NA LINII)----------
");
        Thread.sleep(2000);
    }

    public synchronized void addResource1(int amount) {
        resource1 += amount;
        semaphore1.release(amount);
        System.out.println("Dostarczono Surowiec 1" + ", W ilości: " +
```

```
amount);
        notifyAll();
    }

    public synchronized void addResource2(int amount) {
        resource2 += amount;
        semaphore2.release(amount);
        System.out.println("Dostarczono Surowiec 2" + ", W ilości: " +
amount);
        notifyAll();
    }
    public int getResource1() {
        return resource1;
    }

    public int getResource2() {
        return resource2;
    }

    public int getFinishedProducts() {
        return finishedProducts;
    }
    public synchronized int takeFinishedProducts(int amount) {
        while (finishedProducts < amount) {
            try {
                wait();

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        finishedProducts -= amount;
        notifyAll();
        System.out.println("Brak materialow. POTRZEBNA DOSTAWA!");

        return amount;
    }
}
```

3)

```
class ShippingTruck {
    private ProductionLine line;
    private Random rand;

    public ShippingTruck(ProductionLine line) {
```

```
        this.line = line;
        this.rand = new Random();
    }

    public void run() {
        while (true) {
            int packageAmount = rand.nextInt(100);
            line.takeFinishedProducts(packageAmount);
            int shippingTime = rand.nextInt(100);
            try {
                Thread.sleep(shippingTime);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

4)

```
class SupplyTruck {
    private ProductionLine line;
    private Random rand;
    public SupplyTruck(ProductionLine line) {
        this.line = line;
        this.rand = new Random();
    }
    public void run() {
        while (true) {
            int resource1Amount = rand.nextInt(100);
            int resource2Amount = rand.nextInt(100);
            line.addResource1(resource1Amount);
            line.addResource2(resource2Amount);
            int supplyTime = rand.nextInt(100);
            try {
                Thread.sleep(supplyTime);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```