Teoria współbieżności

Producenci i konsumenci bez zagłodzenia

Jan Gawroński

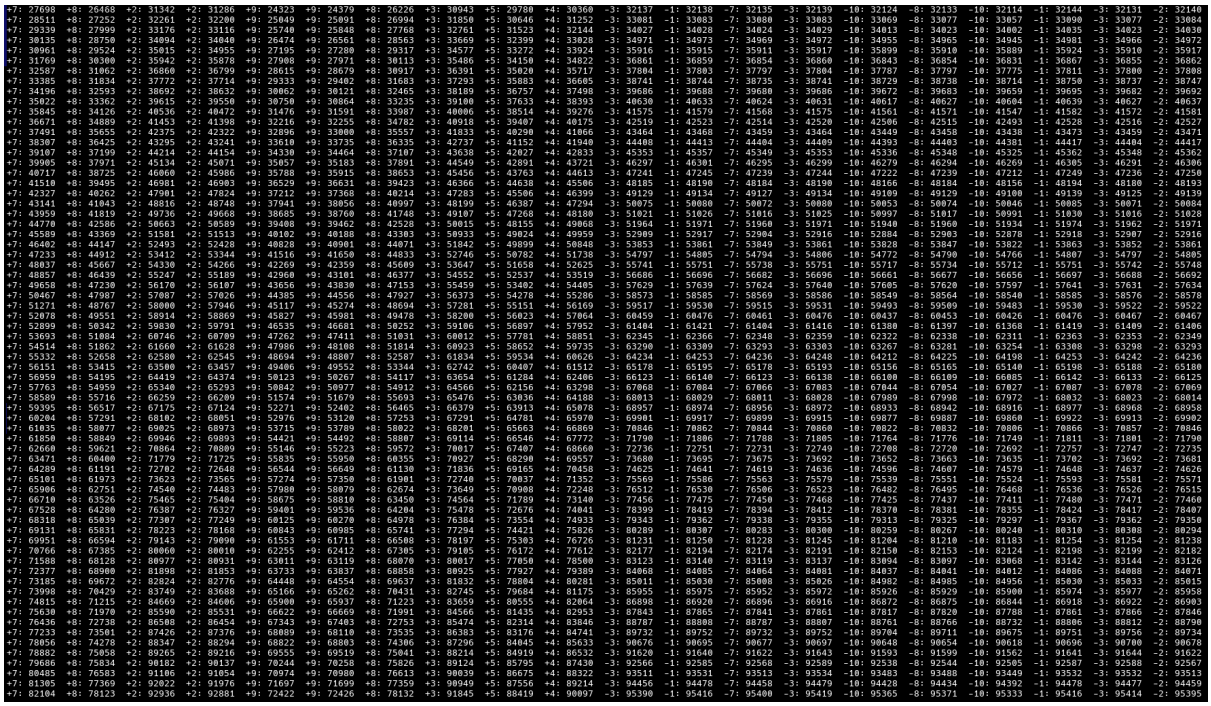
12.11.2025

1 Poprawny kod (4 warunki + zmnienna logiczna)

```java
package agh.ics.concurrency.lab4;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;

public class CorrectRandomProduceConsume {
    private int buffer = 0;
    private final int bufferMax;
    private final ReentrantLock lock = new ReentrantLock();
    private final Condition firstProd = lock.newCondition();
    private final Condition restProd = lock.newCondition();
    private final Condition firstCons = lock.newCondition();
    private final Condition restCons = lock.newCondition();

    private boolean isFirstProducerWaiting = false;
    private boolean isFirstConsumerWaiting = false;

    public CorrectRandomProduceConsume(int bufferMax) {
        this.bufferMax = bufferMax;
    }

    public void produce(int amount) {
        lock.lock();
        try {
            while (isFirstProducerWaiting)
                restProd.await();
            isFirstProducerWaiting = true;
            while (buffer + amount >= bufferMax)
                firstProd.await();
            buffer += amount;
            isFirstProducerWaiting = false;
            restProd.signal();
            firstCons.signal();
```

```
34          } catch (InterruptedException e) {
35              e.printStackTrace();
36          } finally {
37              lock.unlock();
38          }
39      }
40
41      public void consume(int amount) {
42          lock.lock();
43          try {
44              while (isFirstConsumerWaiting)
45                  restCons.await();
46              isFirstConsumerWaiting = true;
47              while (buffer - amount < 0)
48                  firstCons.await();
49              buffer -= amount;
50              isFirstConsumerWaiting = false;
51              restCons.signal();
52              firstProd.signal();
53          } catch (InterruptedException e) {
54              e.printStackTrace();
55          } finally {
56              lock.unlock();
57          }
58      }
59
60      public void print() {
61          System.out.println(this.buffer);
62      }
63
64  }
```

2 Zagłodzenie na 2 warunkach



Jak do tego dochodzi:

bufferMax = 10

1. Bufer jest pełny.
2. Wchodzi P1, który chce wyprodukować 4 i wiesza się na notFull.
3. Wchodzi P2, który chce wyprodukować 1 i wiesza się na notFull.
4. Wchodzi K1, konsumuje 2 i budzi P1.
5. P1 budzi się i wiesza się na notFull.
6. Wchodzi K1, konsumuje 2 i budzi P1.
7. P1 budzi się i produkuje 4.
8. Wchodzi P1, który chce wyprodukować 4 i wiesza się na notFull.

Taka sytuacja może się powtarzać w kółko, prowadząc do zagłodzenia P2.

3 Zakleszczenie i zagłodzenie na 4 warunkach z hasWaiters()

bufferMax = 10

1. K1, który chce skonsumować 3 zawiesza się na firstCons

2. P1 produkuje 1 i wywołuje firstCons budząc K1
3. K2, który chce skonsumować 3 wiesza się na firstCons (Tutaj dochodzi do zagłodzenia, ponieważ K1 może już nigdy nie zostać wybudzony)
4. P1 produkuje 1 i wywołuje firstCons budząc K2
5. K3, który chce skonsumować 3 wiesza się na firstCons
6. K1 wiesza się na firstCons
7. K2 wiesza się na firstCons
8. K4, który chce skonsumować 1 wiesza się na restCons
9. P2 produkuje 5 i wywołuje firstCons budząc K2
10. K2 konsumuje 3 i wywołuje restCons budząc K4
11. K4 wiesza się na restCons
12. P2 produkuje 5 i wywołuje firstCons budząc K3
13. K3 konsumuje 3 i wywołuje restCons budząc K4
14. K4 wiesza się na restCons
15. P1 wiesza się na firstProd
16. P2 - Pp wieszają się na restProd
17. K2 wiesza się na restCons
18. K3 wiesza się na restCons