Sprawozdanie nr 7 - Tomasz Szewczyk

Problem pięciu filozofow

Problem

Problem pięciu filozofów to klasyczny przykład zadania synchronizacji wątków. Ilustracją problemu jest pięciu filozofów, którzy siedzą przy stole i myślą lub jedzą. Aby jeść potrzebują dwóch widelców, niestety widelców jest tylko pięć, więc filozofowie musza współdzielić ten zasób. Problem sprowadza się do znalezienia takiego sposobu synchronizacji aby pięć wątków konkurujących o pięć zasobów mogło współistnieć.

Zadanie

Należalo zaimplementować problem pięciu filozofów na trzy różne sposoby: - a. w najbardziej oczywisty sposób, tak aby zaobserwować zjawisko blokady, - b. rozwiązanie z widelcami podnoszonymi jednocześnie, - c. rozwiązanie z lokajem.

Porównanie wydajności rozwiązań a i b dla miliona cykli: - a: 956 ms - b: 1540 ms

Rozwiązanie trywialne:

```
package tomaszszewczyk.lab7a;

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

public class lab7a {
    public static void main(String[] args) throws InterruptedException {
        Fork fork1 = new Fork();
        Fork fork2 = new Fork();
        Fork fork3 = new Fork();
        Fork fork4 = new Fork();
        Fork fork5 = new Fork();

        Man man1 = new Man(1, fork1, fork2);
        Man man2 = new Man(2, fork2, fork3);
        Man man3 = new Man(3, fork3, fork4);
        Man man4 = new Man(4, fork4, fork5);
        Man man5 = new Man(5, fork5, fork1);

        man1.start();
        man2.start();
```

```
        man3.start();
        man4.start();
        man5.start();

        man1.join();
        man2.join();
        man3.join();
        man4.join();
        man5.join();
    }
}

class Fork {
    private Lock lock = new ReentrantLock();

    public void pick() {
        lock.lock();
    }

    public void release() {
        lock.unlock();
    }
}

class Man extends Thread {
    private int counter = 0;
    private Fork fork1;
    private Fork fork2;
    private int id;

    public Man(int new_id, Fork f1, Fork f2) {
        id = new_id;
        fork1 = f1;
        fork2 = f2;
    }

    public void run() {
        while (true) {
            fork1.pick();
            fork2.pick();

            counter++;
            System.out.println("Filozof: " + id + " jadlem " + counter + " razy");

            fork1.release();
            fork2.release();
```

```
        }
    }
}
```

Rozwiązanie z widelcami podnoszonymi jednocześnie:

```
package tomaszszewczyk.lab7b;

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

public class lab7b {
    public static void main(String[] args) throws InterruptedException {
        Fork fork1 = new Fork();
        Fork fork2 = new Fork();
        Fork fork3 = new Fork();
        Fork fork4 = new Fork();
        Fork fork5 = new Fork();

        Man man1 = new Man(1, fork1, fork2);
        Man man2 = new Man(2, fork2, fork3);
        Man man3 = new Man(3, fork3, fork4);
        Man man4 = new Man(4, fork4, fork5);
        Man man5 = new Man(5, fork5, fork1);

        final long start = System.currentTimeMillis();

        man1.start();
        man2.start();
        man3.start();
        man4.start();
        man5.start();

        man1.join();
        man2.join();
        man3.join();
        man4.join();
        man5.join();

        final long end = System.currentTimeMillis();
        System.out.println("Czas: " + (end - start));
    }
}

class Fork {
```

```
    private Lock lock = new ReentrantLock();

    public void pick() {
        lock.lock();
    }

    public void release() {
        lock.unlock();
    }
}

class Man extends Thread {
    private int counter = 0;
    private Fork fork1;
    private Fork fork2;
    private int id;
    private static Lock lock = new ReentrantLock();

    public Man(int new_id, Fork f1, Fork f2) {
        id = new_id;
        fork1 = f1;
        fork2 = f2;
    }

    public void run() {
        while (true) {
            lock.lock();
            fork1.pick();
            fork2.pick();
            lock.unlock();

            counter++;

            fork1.release();
            fork2.release();

            if(counter > 1000000)
                return;
        }
    }
}
```

Rozwiązanie z lokajem:

```
package tomaszszewczyk.lab7c;
```

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

public class lab7c {
    public static void main(String[] args) throws InterruptedException {
        Waiter w = new Waiter(5);

        Man man1 = new Man(1, 0, 1, w);
        Man man2 = new Man(2, 1, 2, w);
        Man man3 = new Man(3, 2, 3, w);
        Man man4 = new Man(4, 3, 4, w);
        Man man5 = new Man(5, 4, 0, w);

        final long start = System.currentTimeMillis();

        man1.start();
        man2.start();
        man3.start();
        man4.start();
        man5.start();

        man1.join();
        man2.join();
        man3.join();
        man4.join();
        man5.join();

        final long end = System.currentTimeMillis();
        System.out.println("Czas: " + (end - start));
    }
}

class Waiter {
    boolean[] in_use;
    Lock main_lock = new ReentrantLock();
    Condition someone_ends = main_lock.newCondition();

    Waiter(int count) {
        in_use = new boolean[count];
    }

    void please(int a, int b) {
        main_lock.lock();

        while (in_use[a] || in_use[b]) {
```

```
            try {
                someone_ends.await();
            } catch (Exception e) {
                System.out.println("interrupted");
            }
        }

        main_lock.unlock();
    }

    void thanks(int a, int b) {
        main_lock.lock();
        in_use[a] = false;
        in_use[b] = false;
        someone_ends.signal();
        main_lock.unlock();
    }
}

class Man extends Thread {
    private int counter = 0;
    private int fork1;
    private int fork2;
    private int id;
    private Waiter waiter;

    Man(int new_id, int f1, int f2, Waiter w) {
        id = new_id;
        fork1 = f1;
        fork2 = f2;
        waiter = w;
    }

    public void run() {
        while (true) {
            waiter.please(fork1, fork2);

            counter++;

            waiter.thanks(fork1, fork2);

            if(counter == 1000000)
                return;
        }
    }
}
```