Teoria współbieżności

Laboratorium 6

Andrzej Ratajczak

1. Rozwiązanie pierwsze z trywialnymi symetrycznymi filozofami.

Celem pierwszego ćwiczenia było zaimplementować rozwiązanie dla problemu 5 filozofów w trywialny sposób. Najprostszym sposobem są symetryczni filozofowie, którzy biorą zawsze najpierw lewy widelec, później prawy. Następnie jedzą i odkładają sztućce po posiłku. Poniższy kod prezentuje rozwiązanie.

```
class Widelec {
    private ReentrantLock lock = new ReentrantLock();

    public void podnies() {
        lock.lock();
    }
    public void odloz() {
        lock.unlock();
    }
}

class Filozof extends Thread {
    private int _licznik = 0;

    private Widelec leftFork;
    private Widelec rightFork;

    public Filozof(Widelec left, Widelec right) {
        leftFork = left;
        rightFork = right;
    }

    public void run() {
        while (true) {
            leftFork.podnies();
            rightFork.podnies();

            // jedzenie
            ++_licznik;
            if (_licznik % 100 == 0) {
                System.out.println("Filozof: " + Thread.currentThread() +
                        "jadlem " + _licznik + " razy");
            }
            // koniec jedzenia
            leftFork.odloz();
            rightFork.odloz();
        }
    }
}

public class MainA {
    public static void main(String[] args) {
        Widelec[] widelce = new Widelec[5];
```

```
        Filozof[] filozofowie = new Filozof[5];
        widelce[0] = new Widelec();
        for(int i = 0; i < 5; i++) {
            if(i != 4)
            widelce[i+1] = new Widelec();
            filozofowie[i] = new Filozof(widelce[i%5], widelce[(i+1)%5]);
        }

        for(int i = 0; i < 5; i++) {
            filozofowie[i].start();
        }
    }
}
```

Kod źródłowy 1. Trywialna implementacja roziwiązania problemu pięciu filozofów.

Kiedy jednak uruchomimy program dochodzi do zakleszczenia. Dzieje się to dla tego, gdyż możliwa jest konfiguracja, że wszyscy filozofowie podniosą jednocześnie lewy widelec i będą oczekiwali na zwolnienie się prawego. Aby poprawić to rozwiązanie zaimplementuję filozofów tak, aby podnosili oba widelce, w przeciwnym wypadku nie rezerwują żadnego.

2. Implementacja filozofów jednocześnie podnoszących widelce.

```
class Widelec {
    private ReentrantLock lock = new ReentrantLock();

    public boolean podnies() {
        return lock.tryLock();
    }

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

class Filozof extends Thread {
    private int _licznik = 0;

    private Widelec leftFork;
    private Widelec rightFork;

    public Filozof(Widelec left, Widelec right) {
        leftFork = left;
        rightFork = right;
    }

    public void run() {
        while (_licznik < 10000000) {
            if(leftFork.podnies()) {
                if(rightFork.podnies()) {
                    ++_licznik;
                    leftFork.odloz();
                    rightFork.odloz();
                } else {
                    leftFork.odloz();
                }
            }
        }
    }
```

```
}

public class MainB {
    public static void main(String[] args) throws Exception {
        Widelec[] widelce = new Widelec[5];
        Filozof[] filozofowie = new Filozof[5];
        widelce[0] = new Widelec();
        for(int i = 0; i < 5; i++) {
            if(i != 4)
            widelce[i+1] = new Widelec();
            filozofowie[i] = new Filozof(widelce[i%5], widelce[(i+1)%5]);
        }
        long start = System.nanoTime();
        for(int i = 0; i < 5; i++) {
            filozofowie[i].start();
        }
        for(int i = 0; i < 5; i++) {
            filozofowie[i].join();
        }
        long exit = System.nanoTime();
        System.out.println(exit - start);



    }
}
```

Kod źródłowy 2. Implementacja rozwiązania problemu filozofów z jednocześnie podnoszonymi widelcami.

Powyższe rozwiązanie nie prowadzi już do zakleszczeń. Jeżeli któryś z filozofów nie może podnieść obu widelców, to nie przechowuje żadnego na później. Problemem tego rozwiązania jest brak synchronizacji który z wątków jadł. Może to prowadzić do zagłodzenia pozostałych wątków. W celu sprawiedliwego rozłożenia zasobów wprowadzimy dodatkowy mechanizm lokaja, który zagwarantuje sprawiedliwy rozdział.

3. Rozwiązanie problemu filozofów z lokajem.

Wprwadzimy kolejny wątek, który reprezentuje lokaja. Odpowiedzialny jest na zbieranie zamówień na chęć jedzenia oraz dysponowanie widelcami w sprawiedliwy sposób.

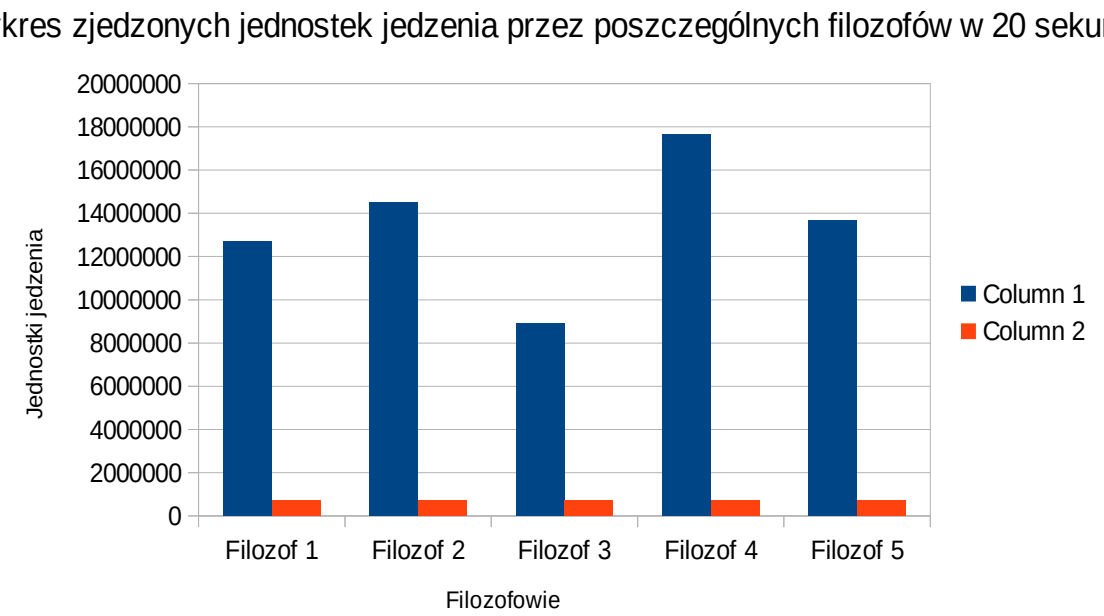
```
class Lokaj extends Thread {
    ReentrantLock lock = new ReentrantLock();
    Queue<Filozof> queue = new ConcurrentLinkedQueue<>();
    AtomicInteger forks = new AtomicInteger(5);

    public void run() {
        while (true) {
            if(!queue.isEmpty() && forks.get() >= 2) {
                lock.lock();
                Filozof filozof = queue.peek();
                queue.remove();
                forks.addAndGet(-2);
                filozof.s.release();
                lock.unlock();
            }
        }
    }

    void dajWidelce(Filozof filozof) {
        lock.lock();
        try {
            filozof.s.acquire();
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
        queue.add(filozof);
        lock.unlock();
    }

    void odbierz(Filozof filozof) {
        lock.lock();
        forks.addAndGet(2);
        filozof.s.release();
        lock.unlock();
    }
}

class Filozof extends Thread {
    Semaphore s = new Semaphore(1);
    private int _licznik = 0;
    Lokaj l;

    Filozof(Lokaj l) {
        this.l = l;
    }

    public void run() {
        while (true) {
            l.dajWidelce(this);
            try {
                s.acquire();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            ++_licznik;
            if (_licznik % 100 == 0) {
                System.out.println("Filozof: " + Thread.currentThread() +
                        "jadlem " + _licznik + " razy");
            }
            l.odbierz(this);
        }
    }
}

public class MainC {
    public static void main(String[] args) throws Exception {
        Lokaj lokaj = new Lokaj();
        Filozof[] filozofowie = new Filozof[5];
        for(int i = 0; i < 5; i++) {
            filozofowie[i] = new Filozof(lokaj);
        }
        long start = System.nanoTime();
        lokaj.start();
        for(int i = 0; i < 5; i++) {
            filozofowie[i].start();
        }
        for(int i = 0; i < 5; i++) {
            filozofowie[i].join();
        }
        long exit = System.nanoTime();
        System.out.println(exit - start);
    }
}
```

Kod źródłowy 3. Implementacja rozwiązania problemu pięciu filozofów z lokajem.

Po zaimplementowaniu zrobiłem testy porównujące drugą oraz trzecią implementację. Kryteriami był czas spożywania przez wszystkich filozofów co najmniej milion jednostek jedzenia oraz sprawiedliwość rozdziału żywności w ciągu 20 sekund działania programu.

	Rozwiązanie 2	Rozwiązanie 3 (Lokaj)
1.000.000 jednostek jedzenia	1.125.375.323 ns	25.494.113.326 ns



Jak widać, pierwszy sposób jest o wiele szybszy, ale nie jest sprawiedliwy i może w krytycznej sytuacji zagłodzić któryś z wątków.