Sprawozdanie nr 5 - Tomasz Szewczyk

Blokowanie drobnoziarniste

Problem

Podczas blokowania przy dostępie do struktur danych zachodzi dodatkowy problem. Czy nie dałoby się uniknąć blokowania dostępu do całej struktury? Czy istnieje sposób umożliwienia dostępu do jednej struktury z wielu wątków pod warunkiem, że każdy wątek będzie korzystał z innej części struktury? Rozwiązaniem jest blokowanie drobnoziarniste. Stosując tą technikę zamykamy zamek jedynie na tym elemencie struktury, z którego aktualnie korzystamy.

Zadanie

Należało zaimplementować listę blokowaną drobnoziarniście oraz blokowaną globalnie, a następnie zmierzyć ich wydajność w zależności od kosztu wykonania operacji.

```
package tomaszszewczyk.lab6;

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

class lab6 {
    public static void main(String[] args) throws InterruptedException {
        class Writer extends Thread {
            private ListIf list;

            public Writer(ListIf aList) {
                list = aList;
            }

            public void run() {
                for (int i = 0; i < 10; i++)
                    list.add(i);
            }
        }

        class Searcher extends Thread {
            private ListIf list;

            public Searcher(ListIf aList) {
                list = aList;
```

```
        }

        public void run() {
            for (int i = 0; i < 10; i++) {
                while (!list.contains(i)) { }
            }
        }
    }

    for(int i = 0; i < 10; i++)
    {
        System.out.println("Koszt: " + i);
        {
            final long start = System.currentTimeMillis();

            List list = new List(-1, i);
            Writer w = new Writer(list);
            Searcher s = new Searcher(list);

            w.start();
            s.start();

            w.join();
            s.join();

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

            ListWithGlobalLock list = new ListWithGlobalLock(-1, i);
            Writer w = new Writer(list);
            Searcher s = new Searcher(list);

            w.start();
            s.start();

            w.join();
            s.join();

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

```
}

interface ListIf {
    boolean contains(Object o);
    ListIf remove(Object o);
    void add(Object o);
}

class List implements ListIf{
    private Object data;
    private List next;
    private Lock lock = new ReentrantLock();
    private int cost;

    public List(Object o, int c) {
        data = o;
        cost = c;
    }

    public List(Object o, int c, List n) {
        data = o;
        next = n;
        cost = c;
    }

    public Object get() {
        return data;
    }

    public List getNext() {
        return next;
    }

    public boolean contains(Object o) {
        boolean result = false;
        lock.lock();
        try {
            Thread.sleep(cost);
            if (o == data) {
                result = true;
            } else if (next != null) {
                result = next.contains(o);
            } else {
                result = false;
            }
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        } finally {
            lock.unlock();
        }

        return result;
    }

    public List remove(Object o) {
        List result = new List(o, cost);
        lock.lock();
        try {
            Thread.sleep(cost);
            if (o == data) {
                result = next;
            } else if (o == next.get()) {
                next = next.getNext();
                result = this;
            } else {
                next.remove(o);
                result = this;
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }

        return result;
    }

    public void add(Object o) {
        lock.lock();
        try {
            Thread.sleep(cost);
            List old_next = next;
            next = new List(o, cost, old_next);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
}

class ListWithGlobalLock implements ListIf {
```

```
private Object data;
private ListWithGlobalLock next;
private static Lock lock = new ReentrantLock();
private int cost;

public ListWithGlobalLock(Object o, int c) {
    data = o;
    cost = c;
}

public ListWithGlobalLock(Object o, int c, ListWithGlobalLock n) {
    data = o;
    next = n;
    cost = c;
}

public Object get() {
    return data;
}

public ListWithGlobalLock getNext() {
    return next;
}

public boolean contains(Object o) {
    boolean result;
    lock.lock();
    result = containsInternal(o);
    lock.unlock();
    return result;
}

private boolean containsInternal(Object o) {
    boolean result;
    try {
        Thread.sleep(cost);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    if (o == data) {
        result = true;
    } else if (next != null) {
        result = next.containsInternal(o);
    } else {
        result = false;
    }
```

```
        return result;
    }

    public ListWithGlobalLock remove(Object o) {
        ListWithGlobalLock result;
        lock.lock();
        result = removeInternal(o);
        lock.unlock();
        return result;
    }

    private ListWithGlobalLock removeInternal(Object o) {
        ListWithGlobalLock result;
        try {
            Thread.sleep(cost);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        if (o == data) {
            result = next;
        } else if (o == next.get()) {
            next = next.getNext();
            result = this;
        } else {
            next.removeInternal(o);
            result = this;
        }
        return result;
    }

    public void add(Object o) {
        lock.lock();
        try {
            Thread.sleep(cost);
            ListWithGlobalLock old_next = next;
            next = new ListWithGlobalLock(o, cost, old_next);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
        }
    }
}
```

Uśrednione wyniki pomiarów:

```
$ ./lab6

Koszt: 0
Blokowanie drobnoziarniste: 13
Blokowanie globalne: 1

Koszt: 1
Blokowanie drobnoziarniste: 15
Blokowanie globalne: 80

Koszt: 2
Blokowanie drobnoziarniste: 18
Blokowanie globalne: 156

Koszt: 3
Blokowanie drobnoziarniste: 23
Blokowanie globalne: 230

Koszt: 4
Blokowanie drobnoziarniste: 30
Blokowanie globalne: 306

Koszt: 5
Blokowanie drobnoziarniste: 38
Blokowanie globalne: 380

Koszt: 6
Blokowanie drobnoziarniste: 45
Blokowanie globalne: 456

Koszt: 7
Blokowanie drobnoziarniste: 53
Blokowanie globalne: 531

Koszt: 8
Blokowanie drobnoziarniste: 60
Blokowanie globalne: 606

Koszt: 9
Blokowanie drobnoziarniste: 68
Blokowanie globalne: 680
```

Zgodnie z oczekekiwaniami blokowanie globalne jest bardziej korzystne tylko wtedy, gdy koszt operacji jest bardzo mały. Im większy koszt wykonanie operacji na liście, tym większa była różnica na niekorzyść blokowania globalnego.