Sprawozdanie nr 2 - Tomasz Szewczyk

Semafor

Problem

Po zaobserwowaniu na pierwszych zajęciach problemu wyścigu poznaliśmy sposoby rozwiązywania problemów tego rodzaju. Semafor to obiekt udostępniający dwie operacje: zajęcie semafora i oddanie semafora. Kiedy jeden wątek zajmie semafor żaden inny nie może tego zrobić do czasu, aż pierwszy odda semafor. Dzięki temu uzyskujemy mechanizm doskonale nadający się do synchronizacji dostępu do zasobu. Semafory po raz pierwszy zostały opisane przez wybitnego badacza Edsgera Djikstrę, który rozwinął w ten sposób algorytm Dekkera.

Zadanie

Na drugich zajęciach należało zaimplementować semafor oraz skorzystać z jego funkcjonalności do usunięcia sytuacji wyścigu z zadania z pierwszych zajęć. Do implementacji semafora zostały wykorzystane wbudowane w język monitory obiektów. Dzięki temu realizacja synchronizacji samego semafora jest realizowana automatycznie.

```
package tomaszszewczyk.lab2;

public class lab2 {
    public static void main(String[] args) throws InterruptedException {
        final long startTime = System.currentTimeMillis();
        final CounterWithSemaphore myCounter = new CounterWithSemaphore();

        Thread firstThread = new Thread() {
            public void run() {
                for (int i = 0; i < 1000000; i++) {
                    myCounter.inc();
                }
            }
        };

        Thread secondThread = new Thread() {
            public void run() {
                for (int i = 0; i < 1000000; i++) {
                    myCounter.dec();
                }
            }
```

```
        };

        firstThread.start();
        secondThread.start();
        secondThread.join();
        firstThread.join();
        final long endTime = System.currentTimeMillis();
        System.out.println(myCounter.getVal());
        System.out.println(endTime - startTime);
    }
}

class CounterWithSemaphore extends Thread {
    private int val;
    private Semaphore sem;

    CounterWithSemaphore() {
        this.val = 0;
        this.sem = new Semaphore();
    }

    void inc() {
        sem.acquire();
        this.val++;
        sem.release();
    }

    void dec() {
        sem.acquire();
        this.val--;
        sem.release();
    }

    int getVal() {
        return val;
    }
}

class Semaphore {

    private SemState state;
    private int waiting;
    Semaphore() {
        this.state = SemState.RELEASED;
        this.waiting = 0;
    }
```

```
    synchronized void acquire() {
        waiting++;
        while (state == SemState.ACQUIRED) {
            try {
                wait();
            } catch (InterruptedException ie) {
                System.out.println(ie.getMessage());
            }
        }
        waiting--;
        state = SemState.ACQUIRED;
    }

    synchronized void release() {
        if (waiting > 0) {
            this.notify();
        }
        state = SemState.RELEASED;
    }

    private enum SemState {ACQUIRED, RELEASED}
}
```

Kilkuktornie uruchomienie programu za każdym razem daje ten sam, poprawny wynik, ale czas trwania operacji znacząco się wydłużył:

```
$ ./lab2
0
1002

$ ./lab2
0
1131

$ ./lab2
0
987
```