19.10.2021r.

Teoria Współbieżności
Sprawozdanie z laboratorium 1.
„Współbieżność w Javie”

Joanna Bryk, grupa środa 17:50

1. Wstęp

Tematem laboratorium 1. były wątki w Javie. Zadanie polegało na napisaniu programu, który uruchamia dwa wątki, które dokonują operacji na tej samej liczbie całkowitej. Następnie należało zaobserwować, jakie wartości przyjmuje ta zmienna na końcu działania programu. Pomocne w interpretacji wyników było wykonanie histogramu. Kolejnym zadaniem było usprawnienie programu, nie korzystając z funkcji systemowych.

2. Rozwiązanie zadań i interpretacja wyników

2.1. W pierwszym zadaniu należało uzupełnić szkielet w taki sposób, aby uruchomić dwa wątki, z których jeden zwiększa wartość zmiennej całkowitej o 1 (klasa IThread), a drugi zmniejsza wartość o 1 (klasa DThread). Na początku wartość zmiennej była 0. Operacje zwiększania i zmniejszania wykonują się po 10000 razy.

Poniżej umieszony został kod programu Race.java:

```
public class Counter {
    private int _val;
    public Counter(int n) {
        _val = n;
    }
    public void inc() {
        _val++;
    }
    public void dec() {
        _val--;
    }
    public int value() {
        return _val;
    }
}
```

```java
// Watek, ktory dekrementuje licznik 100.000 razy
public class DThread extends Thread{
    private Counter cnt;

    public DThread(Counter cnt){
        this.cnt = cnt;
    }

    public void run() {
        for(int i =0;i<10000;i++) {
            cnt.dec();
        }
    }
}
```

```java
// Watek, ktory inkrementuje licznik 100.000 razy
public class IThread extends  Thread {
    private Counter cnt;

    public IThread(Counter cnt){
        this.cnt = cnt;
    }

    public void run() {
        for(int i =0;i<10000;i++) {
            cnt.inc();
        }
    }
}
```

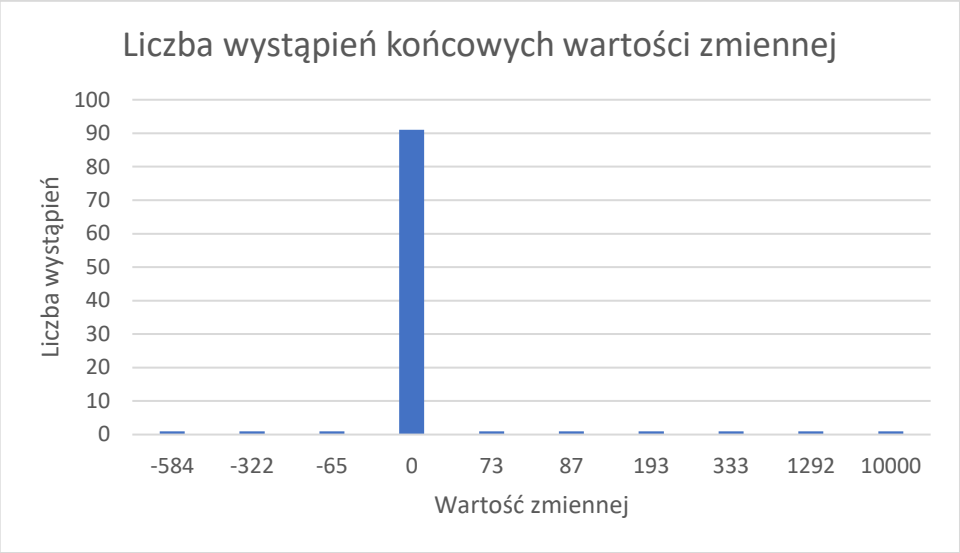
```java
public class Race extends Thread{
    public static void main(String[] args) throws InterruptedException {
        Counter cnt = new Counter(0);

        IThread iThread = new IThread(cnt);
        DThread dThread = new DThread(cnt);

        iThread.start();
        dThread.start();

        iThread.join();
        dThread.join();

        System.out.println("stan=" + cnt.value());
    }
}
```

Program został uruchomiony kilkukrotnie. Za każdym razem wynik końcowy był inny od poprzednich. Spowodowane jest to faktem, że zarówno dodawanie jak i odejmowanie nie są operacjami atomowymi, wykonują się w kilku krokach. Zdarza się, że kiedy jeden wątek nie zdążył zmienić wartości, ten drugi „nie wie” o tym, że będzie ona zmieniona. Tym samym ten z wątków, który pierwszy zakończy swoje działanie nie będzie miał realnego wpływu na wartość zmiennej, ta operacja zostanie zignorowana.

2.2. W celu wykonania drugiego zadania, kod programu musiał zostać lekko zmodyfikowany (poniżej wklejono wprowadzone zmiany).

```
import java.util.*;

public class Race extends Thread {
    public static void main(String[] args) throws InterruptedException {

        HashMap<Integer, Integer> histogram = new HashMap<>();

        for (int i = 0; i < 100; i++) {
            Counter cnt = new Counter(0);
            IThread iThread = new IThread(cnt);
            DThread dThread = new DThread(cnt);
            iThread.start();
            dThread.start();

            iThread.join();
            dThread.join();

            Integer how_many = histogram.get(cnt.value());

            if (how_many == null) {
                histogram.put(cnt.value(), 1);
            } else {
                histogram.put(cnt.value(), histogram.get(cnt.value())
+1);
            }
        }

        Integer[] values = histogram.keySet().toArray(new Integer[0]);
        int[] freq = new int[histogram.size()];
        Arrays.sort(values);

        int i = 0;
        for (Integer j : values) {
            freq[i++] = histogram.get(j);
        }

        for (int k = 0; k < histogram.size(); k++) {
            System.out.println(values[k] + " " + freq[k]);
        }
    }
}
```

Program wykonuje się 100 razy. Uzyskany histogram:



Jak widać, większość końcowych wyników wynosi 0, jednak są wyniki odstające (np. jedno wystąpienie liczby -584).

2.3. W zadaniu trzecim należało usprawnić działanie programu bez pomocy funkcji systemowych. Postanowiłam stworzyć narzędzie, które w swoim działaniu przypomina mutex, blokujący działanie danego wątku w odpowiednim momencie i „uruchamianie" go po wykonaniu konkretnej operacji. W rozwiązaniu tym postanowiłam stworzyć osobną klasę SyncThreads.

```
3. public class SyncThreads {
        private boolean IThreadWorks;
        private boolean DThreadWorks;

        public SyncThreads(){
            this.IThreadWorks = true;
            this.DThreadWorks = false;
        }

        public boolean checkIThread(){
            return  this.IThreadWorks;
        }

        public boolean checkDThread(){
            return this.DThreadWorks;
        }

        public void stopIThread(){
            this.IThreadWorks = false;
        }
        public void stopDThread(){
            this.DThreadWorks = false;
        }

        public void startIThread(){
            this.IThreadWorks = true;
```

```
        }
        public void startDThread(){
            this.DThreadWorks = true;
        }
    }
```

Następnie udoskonaliłam metodę run w obu klasach.

```
public void run() {
    for(int i =0;i<10000;i++) {
        if (!this.sync.checkDThread()){
            System.out.println("DTHREAD " + i + " " + this.cnt.value());
        }
        else {
            cnt.dec();
            this.sync.stopDThread(); //zmieniam na falsz
            this.sync.startIThread(); //zaczynam ten drugi watek
        }
    }
}
```

Rozwiązanie to jednak jest niepoprawne. Inne sposoby, które próbowałam także nie przyniosły oczekiwanego skutku.

4. Podsumowanie

Ten prosty program ukazał, że wątki, które nie zostaną zsynchronizowane zachowują się w nieprzewidywalny, niedeterministyczny sposób. Własność ta ukazana została na histogramie. Z tego powodu należy stosować mechanizmy, które synchronizują działanie wątków. Podjęta w zadaniu 3. próba zsynchronizowania wątków zakończyła się niepowodzeniem. Należy jednak pamiętać, że język Java posiada własne, systemowe narzędzia, które usprawniają działanie programów wielowątkowych.