Laboratorium I – Teoria współbieżności

Weronika Szybińska, 17.10.2022

Treść zadania:

1. Napisać program, który uruchamia 2 wątki, z których jeden zwiększa wartość zmiennej całkowitej o 1, drugi wątek zmniejsza wartość o 1. Zakładając że na początku wartość zmiennej Counter była 0, chcielibyśmy wiedzieć jaka będzie wartość tej zmiennej po wykonaniu 10000 operacji zwiększania i zmniejszania przez obydwa wątki.
2. Na podstawie 100 wykonań programu z p.1, stworzyć histogram końcowych wartości zmiennej Counter.
3. Spróbować wprowadzić mechanizm do programu z p.1, który zagwarantowałby przewidywalną końcową wartość zmiennej Counter. Nie używać żadnych systemowych mechanizmów, tylko swój autorski.

Opis rozwiązania oraz wyniki w formie histogramu:

1. Do wykonania zadania pierwszego użyty został szkielet dostarczony wraz z treścią zadania. Trzy z czterech klas znajdujących się w podanym programie (class IThread, class DThread oraz class Race) zostały zmodyfikowane.

```
20    // Watek, ktory inkrementuje licznik 100.000 razy
21    class IThread extends Thread {
22        public Counter cnt;
23
24        public IThread(Counter cnt){this.cnt = cnt;}
25
26        public void run() {
27            for (int i=0; i<100000; i++){
28                cnt.inc();
29            }
30        }
31    }
```

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

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

    public void run() {
        for (int i=0; i<100000; i++){
            cnt.dec();
        }
    }
}
```

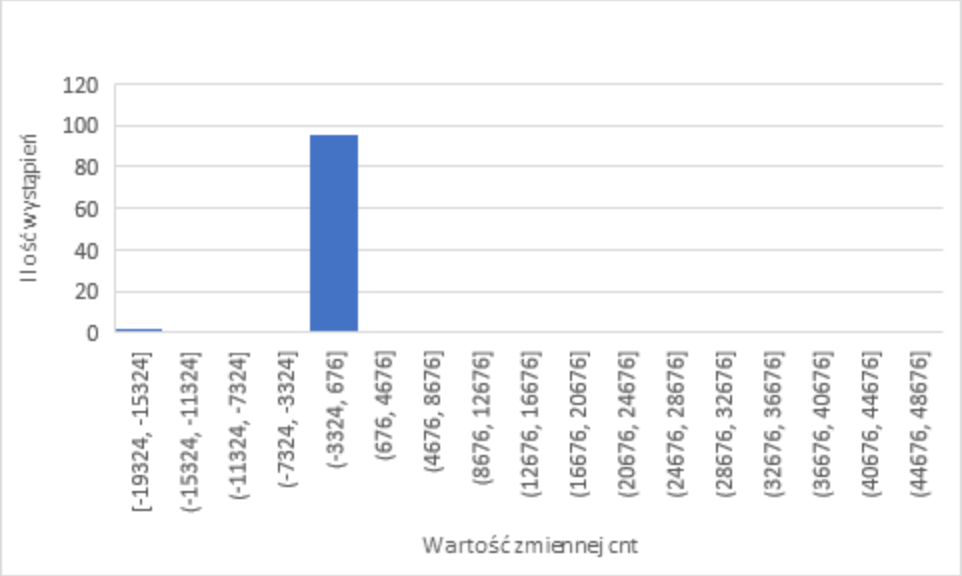
```java
public class Race {
    public static void main(String[] args) throws InterruptedException, IOException {
        Counter cnt = new Counter( n: 0);
        IThread incThread = new IThread(cnt);
        DThread decThread = new DThread(cnt);

        incThread.start();
        decThread.start();

        incThread.join();
        decThread.join();

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

Po modyfikacji i puszczeniu programu można było zauważyć, że wartość zmiennej cnt modyfikowanej przez oba wątki, po zakończeniu programu były różne od 0.

2. Klasa Race została następnie zmodyfikowana, aby móc stworzyć histogram końcowych wartości zmiennej cnt po 100 wykonaniach programu.

```
45  public class Race {
46      public static void main(String[] args) throws InterruptedException, IOException {
47          File newFile = new File( pathname: "histogram2.csv");
48
49          for(int i=0; i<100; i++) {
50              Counter cnt = new Counter( n: 0);
51              IThread incThread = new IThread(cnt);
52              DThread decThread = new DThread(cnt);
53
54              incThread.start();
55              decThread.start();
56
57              incThread.join();
58              decThread.join();
59
60              try {
61                  FileWriter outputfile = new FileWriter(newFile, append: true);
62                  BufferedWriter bw = new BufferedWriter(outputfile);
63                  PrintWriter pw = new PrintWriter(bw);
64                  pw.println(Integer.toString(cnt.value()));
65                  pw.flush();
66                  pw.close();
67              } catch (IOException e) {
68                  e.printStackTrace();
69              }
70          }
71      }
72  }
```

Poniższy obrazek przedstawia powstały histogram.



Analiza owego histogramu daję nam podstawy, aby uważać, że w przypadku wielkokrotnego puszczenia programu, wartość zmiennej cnt po zakończeniu programu jest w znacznej większości przypadków równa lub bliska 0.

3. Aby zagwarantować przewidywalną wartość końcową zmiennej cnt, zmodyfikowane zostały klasy IThread oraz DThread przy użyciu metod Thread.sleep() oraz Thread.interrupt(). Oba wątki za pomocą owych metod usypiają się i wzajemnie wybudzają w taki sposób, aby w danym momencie tylko jedna z nich modyfikowała zmienną cnt. Dzięki takiemu rozwiązaniu końcowa wartość zmiennej jest zawsze równa 0, lecz jest to nieoptymalne rozwiązanie, które znacząco zwiększa czas działania programu.

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

    public IThread(Counter cnt){
        this.cnt = cnt;
    }
    public void assignOpposite(Thread opposite){
        this.opposite = opposite;
    }

    public void run() {
        for (int i=0; i<1000; i++){
            cnt.inc();
            opposite.interrupt();
            try {
                sleep( millis: 1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        if(!opposite.isInterrupted()){
            opposite.interrupt();
        }
    }

}
```

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

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

    public void assignOpposite(Thread opposite){
        this.opposite = opposite;
    }

    public void run() {
        for (int i=0; i<1000; i++) {
            cnt.dec();
            opposite.interrupt();
            try {
                sleep( millis: 1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        if(!opposite.isInterrupted()){
            opposite.interrupt();
        }
    }
}
```

Wnioski:

W trakcie zadania ukazał się problem powszechnie nazywany race condition. Zachodzi on w sytuacji, gdy kilka wątków (w omawianym przypadku dwa) modyfikuje tą samą zmienną. Sytuacja taka może prowadzić do powstania trudnych do wykrycia błędów. Przy rozwiązywaniu zadania 1 oraz 2 pokazane zostało, że wartości końcowe takiej zmiennej nie zawsze były równe 0, pomimo zaimplementowania takiej samej ilości akcji inkrementacji i dekrementacji. Do wykluczenia możliwości wystąpienia takiego problemu wykorzystany został prosty, lecz nieoptymalny sposób usypiania i wybudzania wątków.

Bibliografia:

1. https://home.agh.edu.pl/~funika/tw/lab1/
2. https://biegajacyprogramista.pl/2021/09/27/watki-wielowatkowosc-oraz-wyscig-race-condition/
3. https://developeronthego.pl/java-watki-przetwarzanie-wielowatkowe/
4. https://www.samouczekprogramisty.pl/watki-w-jezyku-java/