SPRAWOZDANIE VI

TEORIA WSPÓŁBIEŻNOŚCI

Problem pięciu filozofów

DAWID BIAŁKA

DATA LABORATORIUM 10.11.2020

DATA ODDANIA 17.11.2020

- Ćwiczenia (Szkielet programu)
    a. Zaimplementować trywialne rozwiązanie z symetrycznymi filozofami. Zaobserwować problem blokady.
    b. Zaimplementować rozwiazanię z widelcami podnoszonymi jednocześnie. Jaki problem może tutaj wystąpić ?
    c. Zaimplementować rozwiązanie z lokajem.
    d. Wykonać pomiary dla każdego rozwiązania i wywnioskować co ma wpływ na wydajność każdego rozwiązania

Zad a.

Koncepcja

Mamy pięciu filozofów i pięć widelców. Każdy filozof może podnieść lewy widelec nie sprawdzając, czy może podnieść prawy. Może dojść do sytuacji, gdzie wszyscy filozofowie podniosą swój lewy widelec i żaden z nich nie będzie mógł podnieść prawego (bo wszystkie są już zajęte). Na podstawie szkieletu zamieszczonego na stronie implementujemy tę sytuację.

Implementacja i wyniki

```
public class Main {

    public static void main(String[] args) {
        ArrayList<Filozof> filozofowie = new ArrayList<>();
        TimeContainer timeContainer = new TimeContainer();
        ArrayList<Widelec> widelce = new ArrayList<>();

        for(int i=0; i<5; i++) {
            widelce.add(new Widelec());
        }

        for(int i=0; i<5; i++) {
            Filozof filozof = new Filozof(i, timeContainer, widelce);
            filozofowie.add(filozof);
            filozof.start();
        }

        for(int i=0; i<5; i++) {
            try {
                filozofowie.get(i).join();
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

    }
}
```

```
public class TimeContainer {
   private long waitingTime = 0;

   public TimeContainer() {}

   public void updateWaitingTime(long start, long end) {
      this.waitingTime += (end - start);
   }

   public long getWaitingTime() {
      return this.waitingTime / 1000000;
   }

}
```

```
public class Widelec {
   boolean podniesiony = false;

   public synchronized void podnies() {
      try{
         while(podniesiony)
         {
            wait();
         }
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
      podniesiony = true;

   }
   public synchronized void odloz() {
      podniesiony = false;
      notify();
   }
}
```

```java
public class Filozof extends Thread {
    private int _licznik = 0;
    private int ID = 0;
    private TimeContainer timeContainer;
    private ArrayList<Widelec> widelce;

    public Filozof(int ID, TimeContainer timeContainer, ArrayList<Widelec> widelce){
        this.ID = ID;
        this.timeContainer = timeContainer;
        this.widelce = widelce;
    }

    public void run() {
        int lewyWidelec = ID;
        int prawyWidelec = (ID+1)%5;
        while (true) {

            long start = System.nanoTime();

            widelce.get(lewyWidelec).podnies();
            System.out.println("Filozof: " + this.ID + " podniosłem lewy widelec");
            widelce.get(prawyWidelec).podnies();
            System.out.println("Filozof: " + this.ID + " podniosłem prawy widelec i zaczynam jeść");

            ++_licznik;
            if (_licznik % 1000 == 0) {
                System.out.println("Filozof: " + this.ID +
                        "jadlem " + _licznik + " razy");
            }

            widelce.get(lewyWidelec).odloz();
            System.out.println("Filozof: " + this.ID + " odłożyłem lewy widelec");
            widelce.get(prawyWidelec).odloz();
            System.out.println("Filozof: " + this.ID + " odłożyłem prawy widelec i skończyłem jeść");
            timeContainer.updateWaitingTime(start, System.nanoTime());

        }
    }
}
```

Po uruchomieniu programu widzimy, że wszyscy filozofowie podnoszą swój lewy widelec i program się zakleszcza.

```
Filozof: 3 podniosłem lewy widelec
Filozof: 1 podniosłem lewy widelec
Filozof: 0 odłożyłem prawy widelec i skończyłem jeść
Filozof: 2 podniosłem lewy widelec
Filozof: 4 podniosłem lewy widelec
Filozof: 0 podniosłem lewy widelec
```

Wnioski

Rozwiązanie to nie jest poprawne, ponieważ program zakleszcza się. Należy wprowadzić modyfikację do programu, co zrobimy w punkcie b.

Zad b.

Koncepcja

Tutaj sytuacja jest taka jak poprzednio, jednakże w jednym czasie tylko jeden filozof może próbować podnieść widelce i musi za jednym razem podnieść dwa. Aby osiągnąć taki warunek wykorzystamy semafor, który będzie podnoszony przez filozofa przez próbą podniesienia widelców. Jeśli ktoś jest w trakcie podnoszenia widelców lub czeka na któryś z nich to wtedy żaden filozof nie może próbować podnieść widelców i czeka na semaforze.

Implementacja i wyniki

```
public class Main {

   public static void main(String[] args) {
      ArrayList<Filozof> filozofowie = new ArrayList<>();
      ArrayList<Widelec> widelce = new ArrayList<>();
      Semaphore sem = new Semaphore(1);

      for(int i=0; i<5; i++) {
         widelce.add(new Widelec());
      }

      for(int i=0; i<5; i++) {
         Filozof filozof = new Filozof(i, new TimeContainer(), widelce, sem);
         filozofowie.add(filozof);
         filozof.start();
      }

      for(int i=0; i<5; i++) {
         try {
            filozofowie.get(i).join();
         }
         catch (InterruptedException e) {
            e.printStackTrace();
         }
      }
   }
}
```

```java
public class TimeContainer {
    private long waitingTime = 0;

    public TimeContainer() {}

    public void updateWaitingTime(long start, long end) {
        this.waitingTime += (end - start);
    }

    public long getWaitingTime() {
        return this.waitingTime / 1000000;
    }

    public void saveResults(int numberOfEatings) {
        try {
            FileWriter myWriter = new FileWriter("results.txt", true);
            myWriter.write(String.format("%d %d", numberOfEatings, getWaitingTime()) + "\n");
            myWriter.close();
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

```java
public class Widelec {
    boolean podniesiony = false;

    public synchronized void podnies() {
        try{
            while(podniesiony)
            {
                wait();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        podniesiony = true;

    }
    public synchronized void odloz() {
        podniesiony = false;
        notify();
    }
}
```

```
public class Filozof extends Thread {
   private int _licznik = 0;
   private int ID = 0;
   private final TimeContainer timeContainer;
   private final ArrayList<Widelec> widelce;
   private final Semaphore sem;

   public Filozof(int ID, TimeContainer timeContainer, ArrayList<Widelec> widelce, Semaphore sem){
      this.ID = ID;
      this.timeContainer = timeContainer;
      this.widelce = widelce;
      this.sem = sem;
   }

   public void run() {
      int lewyWidelec = ID;
      int prawyWidelec = (ID+1)%5;
      while (true) {

         long start = System.nanoTime();

         try {
            sem.acquire(1);

            widelce.get(lewyWidelec).podnies();
            widelce.get(prawyWidelec).podnies();

            sem.release(1);
         } catch (InterruptedException e) {
            e.printStackTrace();
         }

         ++_licznik;

         widelce.get(lewyWidelec).odloz();
         widelce.get(prawyWidelec).odloz();

         timeContainer.updateWaitingTime(start, System.nanoTime());

         if (_licznik % 100000 == 0) {
            System.out.println("Filozof: " + this.ID +
                  " jadlem " + _licznik + " razy");
            timeContainer.saveResults(_licznik);
         }
      }
   }
}
```

Pomiary czasu wykonania się programu filozofa zostały wykonane do momentu, gdy filozof łącznie zjadł 1000000 razy. Pomiary te będą porównane w następnym zadaniu z pomiarami dla wersji z lokajem.

Wnioski

W tym przypadku program nie zakleszcza się. Warunek, że na raz trzeba podnieść dwa widelce gwarantuje nam działanie programu bez zakleszczenia.

Występujący tu problem wiąże się z efektywnością. Gdy jeden z filozofów podniesie semafor (który w tym momencie był opuszczony) i będzie chciał uzyskać widelce, ale jeden z nich jest w tym momencie zajęty przez innego filozofa, to chcący podnieść widelce filozof będzie musiał czekać na zwolnienie tych widelców cały czas trzymając semafor. W tym czasie inni filozofowie, których widelce są wolne i chcieliby je podnieść nie mogą tego zrobić bo semafor jest podniesiony i muszą czekać.

Zad c.

Koncepcja

Tutaj dopuszczamy do stołu na raz tylko czterech filozofów. Zajmuje się tym lokaj (semafor z wartością początkową 4). Jeśli filozof chce podejść do stołu a siedzi przy nim już czterech filozofów, którzy jedzą, to musi poczekać, aż ktoś skończy jeść i odejdzie od stołu.

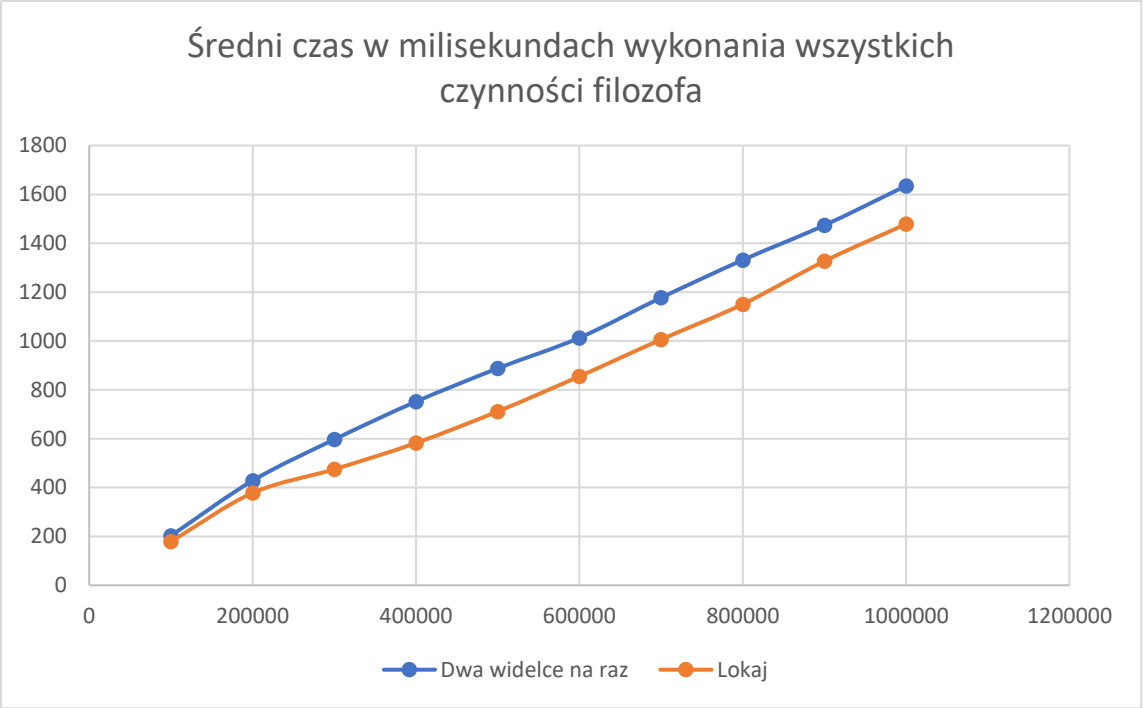
Implementacja i wyniki

```
public class Main {

   public static void main(String[] args) {
      ArrayList<Filozof> filozofowie = new ArrayList<>();
      ArrayList<Widelec> widelce = new ArrayList<>();
      Semaphore sem = new Semaphore(4);

      for(int i=0; i<5; i++) {
         widelce.add(new Widelec());
      }

      for(int i=0; i<5; i++) {
         Filozof filozof = new Filozof(i, new TimeContainer(), widelce, sem);
         filozofowie.add(filozof);
         filozof.start();
      }

      for(int i=0; i<5; i++) {
         try {
            filozofowie.get(i).join();
         }
         catch (InterruptedException e) {
            e.printStackTrace();
         }
      }
   }
}
```

```java
public class TimeContainer {
    private long waitingTime = 0;

    public TimeContainer() {}

    public void updateWaitingTime(long start, long end) {
        this.waitingTime += (end - start);
    }

    public long getWaitingTime() {
        return this.waitingTime / 1000000;
    }

    public void saveResults(int numberOfEatings) {
        try {
            FileWriter myWriter = new FileWriter("results.txt", true);
            myWriter.write(String.format("%d %d", numberOfEatings, getWaitingTime()) + "\n");
            myWriter.close();
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

```java
public class Widelec {
    boolean podniesiony = false;

    public synchronized void podnies() {
        try{
            while(podniesiony)
            {
                wait();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        podniesiony = true;

    }
    public synchronized void odloz() {
        podniesiony = false;
        notifyAll();
    }
}
```

```java
public class Filozof extends Thread {
    private int _licznik = 0;
    private int ID = 0;
    private final TimeContainer timeContainer;
    private final ArrayList<Widelec> widelce;
    private final Semaphore sem;

    public Filozof(int ID, TimeContainer timeContainer, ArrayList<Widelec> widelce, Semaphore sem){
        this.ID = ID;
        this.timeContainer = timeContainer;
        this.widelce = widelce;
        this.sem = sem;
    }

    public void run() {
        int lewyWidelec = ID;
        int prawyWidelec = (ID+1)%5;
        while (true) {

            long start = System.nanoTime();

            try {
                sem.acquire(1);

                widelce.get(lewyWidelec).podnies();
                widelce.get(prawyWidelec).podnies();

                ++_licznik;

                widelce.get(lewyWidelec).odloz();
                widelce.get(prawyWidelec).odloz();

                timeContainer.updateWaitingTime(start, System.nanoTime());

                if (_licznik % 100000 == 0) {
                    System.out.println("Filozof: " + this.ID +
                            " jadlem " + _licznik + " razy");
                    timeContainer.saveResults(_licznik);
                }

                sem.release(1);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```



Wykres średniego czasu w milisekundach wykonania wszystkich czynności filozofa (czas zaczynamy liczyć na samym początku, przed próbą podniesienia semafora i kończymy go liczyć po odłożeniu widelców) w zależności od ogólnej liczby zjedzeń dla wersji z lokajem i podnoszeniem dwóch widelców na raz.

Wnioski

Jeśli jednocześnie przy stole znajduje się tylko czterech filozofów to nie może wystąpić zjawisko zakleszczenia. Z powyższego wykresu widzimy, że wersja z lokajem jest bardziej wydajna niż wersja z podnoszeniem dwóch widelców na raz.

Bibliografia

Z. Weiss, T. Gruźlewski, Programowanie współbieżne i rozproszone. WNT, Warszawa 1993.

http://home.agh.edu.pl/~funika/tw/lab6/

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html

https://pl.wikipedia.org/wiki/Problem_ucztuj%C4%85cych_filozof%C3%B3w [Rozwiązanie przy użyciu kelnera (lokaja)]