Sprawozdanie

Teoria Współbieżności

Laboratorium 6

Joanna Bryk, środa 17:50

1. Wstęp

   Laboratorium szóste polegało na rozwiązaniu problemu pięciu filozofów. Jest to zadanie, w którym należy zsynchronizować zachowanie filozofów siedzących przy okrągłym stole. Ich głównym zajęciem jest myślenie, jednak czasami głodnieją i wtedy, aby zjeść, muszą mieć do dyspozycji oba widelce – po prawej i po lewej ich stronie. Filozof podnosi dwa widelce (uniemożliwiając korzystanie z nich swoim sąsiadom), je i odkłada widelce.
   Zadanie wymagało trzech implementacji:
   - Rozwiązanie trywialne z symetrycznymi filozofami i z zaobserwowaniem problemu blokady.
   - Rozwiązanie z widelcami podnoszonymi jednocześnie.
   - Rozwiązanie z lokajem.
2. Implementacja i obserwacje

   2.1. Implementacja rozwiązania trywialnego:

```
public class Widelec {
    private boolean podniesiony = false;
    public Widelec(){

    }
    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;
    private ArrayList<Widelec> widelce;
    private long czas;

    public Filozof(int ID, ArrayList<Widelec> widelce) {
        this.id = ID;
```

```
        this.widelce = widelce;
    }

    public void run() {
        while (true) {
            Widelec prawy = widelce.get(id);
            Widelec lewy = widelce.get((id + 1) % 5);

            prawy.podnies();
            System.out.println("Tutaj fil " + id + " podniosłem widelec
prawy");
            lewy.podnies();
            System.out.println("Tutaj fil " + id + " podniosłem widelec
lewy");

            // jedzenie
            ++_licznik;
            if (_licznik % 1000 == 0) {
                System.out.println("Filozof: " + Thread.currentThread()
+
                        "jadlem " + _licznik + " razy");
            }
            // koniec jedzenia

            prawy.odloz();
            lewy.odloz();

        }
    }

    public int getID() {
        return this.id;
    }
}
public class Fil5mon {
    public static void main (String[] args) throws InterruptedException
{

        int n = 5;

        ArrayList<Filozof> filozofowie = new ArrayList<>();
        ArrayList<Widelec> widelce = new ArrayList<>();

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

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

        for (Filozof f : filozofowie)
            f.start();

        for (Filozof f : filozofowie)
            f.join();

    }
}
```

Zgodnie z przewidywaniami program zakleszcza się – każdy filozof podnosi swój widelec, tylko jeden, żaden z nich nie może zjeść.

2.2. Implementacja z jednoczesnym podnoszeniem widelców polega na dodaniu semafora, który pilnuje, aby tylko jeden filozof podnosił widelce i aby na pewno podniósł dwa. Zmianie uległa klasa Filozof. W klasie Fil5mon tworzony jest semafor (z początkową wartością 1), który następnie przekazuję jako argument każdemu z filozofów. Dodałam również metodę *Czekanie()* potrzebną do wykonania wykresu.

```
public class Filozof extends Thread{
    private int _licznik = 0;
    private int id;
    private ArrayList<Widelec> widelce;
    private long czas;
    private long czasCzekania = 0;
    private Semaphore semafor;

    public Filozof(int ID, ArrayList<Widelec> widelce, Semaphore
semafor){
        this.id = ID;
        this.widelce = widelce;
        this.semafor = semafor;
    }

    public void run() {
        while (true) {
            Widelec prawy = widelce.get(id);
            Widelec lewy = widelce.get((id + 1) % 5);

            long start = System.nanoTime();
            try {
                semafor.acquire(1);
                prawy.podnies();
                lewy.podnies();
                semafor.release(1);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            // jedzenie
            ++_licznik;

            // koniec jedzenia

            prawy.odloz();
            lewy.odloz();

            long t = System.nanoTime();
            Czekanie((t-start) / 1000000);

            if (_licznik % 100000 == 0) {
                System.out.println(_licznik + " " + czasCzekania);
            }
        }
    }

    public void Czekanie(long t){
        this.czasCzekania +=t;
```

```
    }
}
```

Co ważne, w tej implementacji program nie zakleszcza się. Jednak nie jest to wydajne rozwiązanie, jako że wszyscy inni filozofowie czekają w czasie, kiedy tylko jeden z nich może jeść.

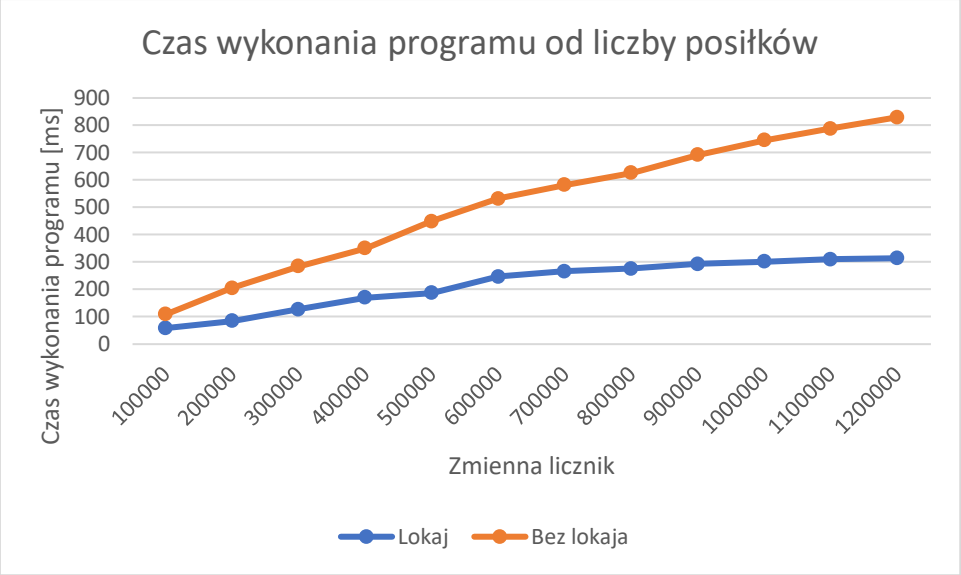
2.3. Ostatnia implementacja rożni się od poprzednich lokajem – arbitrem, który pilnuje, aby w każdej chwili co najwyżej czterech filozofów próbowało podnieść widelce. Rozwiązanie takie, jest poprawne – wyklucza zjawisko blokady i zagłodzenie filozofa. Lokaj jest zaimplementowany jako semafor z wartością początkową 4.

```
public class Fil5mon{
        public static void main(String[] args) throws
InterruptedException {
        int n = 5;

        ArrayList<Filozof> filozofowie = new ArrayList<>();
        ArrayList<Widelec> widelce = new ArrayList<>();

        Semaphore semafor = new Semaphore(4);

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

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

        for (Filozof f : filozofowie)
            f.start();

        for (Filozof f : filozofowie)
            f.join();
    }
}
public class Filozof extends Thread {
    private int _licznik = 0;
    private int id;
    private final ArrayList<Widelec> widelce;
    private long czas;
    private long czasCzekania = 0;
    private final Semaphore semafor;

    public Filozof(int ID, ArrayList<Widelec> widelce, Semaphore
semafor) {
        this.id = ID;
        this.widelce = widelce;
        this.semafor = semafor;
    }

    public void run() {
        Widelec prawy = widelce.get(id);
        Widelec lewy = widelce.get((id + 1) % 5);
        while (true) {
            long start = System.nanoTime();
```

```
            try {
                semafor.acquire(1);
                prawy.podnies();
                lewy.podnies();

                // jedzenie
                ++_licznik;

                // koniec jedzenia

                prawy.odloz();
                lewy.odloz();

                long t = System.nanoTime();
                Czekanie((t - start) / 1000000);

                if (_licznik % 100000 == 0) {
                    System.out.println(_licznik + " " + czasCzekania);
//                  System.out.println("Filozof: " +
Thread.currentThread() +
//                          "jadlem " + _licznik + " razy, czas czkeania +
" + czasCzekania);
                }

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


        }
    }

    public void Czekanie(long t) {
        this.czasCzekania += t;
    }
}
```

3. Wykres

Wykonany wykres jest porównaniem czasów wykonywania się programów – wersji z lokajem i bez lokaja. Czas wykonywania programu liczony był jako średnia pięciu czasów.



Rysunek 1 Porównanie rozwiązań

4. Podsumowanie

Pierwsza implementacja jest niepoprawna – zachodzi zjawisko blokady, żaden z filozofów nie ma dwóch widelców, żaden nie może zjeść. Rozwiązanie drugie jest lepsze, jednak najlepsze okazuje się rozwiązanie wykorzystujące lokaja. Zgodnie z przewidywaniami, rozwiązanie trzecie jest bardziej wydajne, szybsze niż to bez lokaja. Zależność ta widoczna jest na przedstawionym wyżej wykresie (Rysunek 1.)

5. Bibliografia

- https://home.agh.edu.pl/~funika/tw/lab6/
- Z. Weiss, T. Gruźlewski, *Programowanie współbieżne i rozproszone w przykładach i zadaniach*, WNT, 1993