

Teoria Współbieżności - Sprawozdanie 5

Maciej Brzeżawski

Listopad 2024

1 Treść zadania

Problem pięciu filozofów:

- Każdy filozof zajmuje się głównie myśleniem
- Od czasu do czasu potrzebuje zjeść
- Do jedzenia potrzebne mu są oba widelce po jego prawej i lewej stronie
- Jedzenie trwa skończona (ale nieokreślona z góry) ilość czasu, po czym filozof widelce odkłada i wraca do myślenia
- Cykl powtarza się od początku

Ćwiczenia:

1. Zaimplementować trywialne rozwiązanie z symetrycznymi filozofami. Z Obserwować problem blokady.
2. Zaimplementować rozwiązanie z widelcami podnoszonymi jednocześnie. Jaki problem może tutaj wystąpić?
3. Zaimplementować rozwiązanie z lokajem.
4. Wykonać pomiary dla każdego rozwiązania i wywnioskować co ma wpływ na wydajność każdego rozwiązania

2 Rozwiązanie z symetrycznymi filozofami

2.1 Implementacja programu

```
class Widelec {
    private boolean zajety = false;

    public void podnies() {
        while (zajety) {
            // Aktywne czekanie, co prowadzi do możliwego deadlocku
        }
        zajety = true;
    }

    public void odloz() {
        zajety = false;
    }
}

class Filozof extends Thread {
    private int _licznik = 0;
    private final Widelec lewyWidelec;
    private final Widelec prawyWidelec;

    public Filozof(Widelec lewy, Widelec prawy) {
        this.lewyWidelec = lewy;
        this.prawyWidelec = prawy;
    }

    public void run() {
        while (true) {
            // Filozof myśli
            System.out.println(Thread.currentThread().getName() + " myśli...");
            try {
                Thread.sleep((int) (Math.random() * 100));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            // Filozof próbuje podnieść oba widelce
            lewyWidelec.podnies();
            System.out.println(Thread.currentThread().getName() +
                " podniósł lewy widelec.");

            prawyWidelec.podnies();
            System.out.println(Thread.currentThread().getName() +
```

```

        " podniósł prawy widelec.");

        // Filozof je
        _licznik++;
        System.out.println(Thread.currentThread().getName() +
            " je. Licznik: " + _licznik);
        try {
            Thread.sleep((int) (Math.random() * 100));
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Filozof odkłada widelce
        lewyWidelec.odloz();
        prawyWidelec.odloz();
        System.out.println(Thread.currentThread().getName() +
            " odłożył oba widelce.");
    }
}

public class Fil5mon {
    public static void main(String[] args) {
        int liczbaFilozofow = 5;
        Widelec[] widelce = new Widelec[liczbaFilozofow];
        Filozof[] filozofowie = new Filozof[liczbaFilozofow];

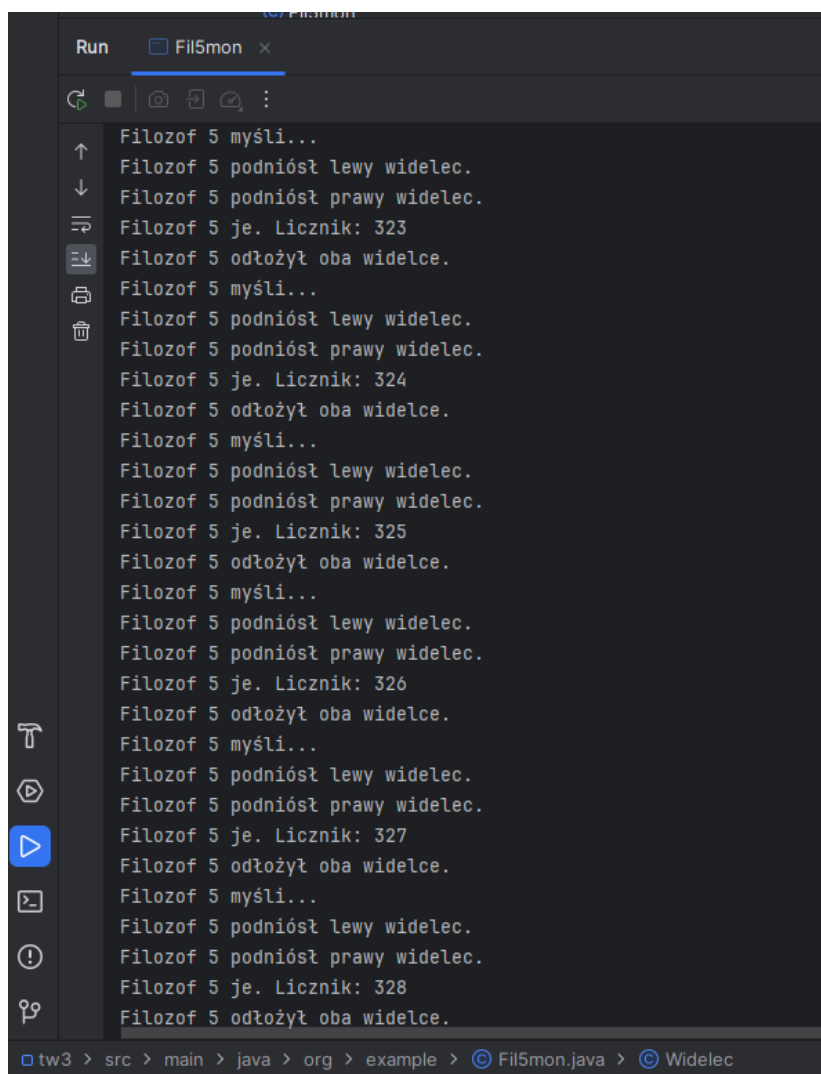
        // Inicjalizacja widelców
        for (int i = 0; i < liczbaFilozofow; i++) {
            widelce[i] = new Widelec();
        }

        // Inicjalizacja filozofów
        for (int i = 0; i < liczbaFilozofow; i++) {
            Widelec lewy = widelce[i];
            Widelec prawy = widelce[(i + 1) % liczbaFilozofow];
            filozofowie[i] = new Filozof(lewy, prawy);
            filozofowie[i].setName("Filozof " + (i + 1));
            filozofowie[i].start();
        }
    }
}

```

2.2 Obserwacja problemu blokady

Symetryczne podejście ilustruje problem blokady, gdyż wszyscy filozofowie mogą jednocześnie podnieść swoje lewe widelce i czekać na prawy. Żaden z filozofów nie może kontynuować, co powoduje zablokowanie programu.



```
Run Fil5mon x
Filozof 5 myśli...
Filozof 5 podniósł lewy widelec.
Filozof 5 podniósł prawy widelec.
Filozof 5 je. Licznik: 323
Filozof 5 odłożył oba widelce.
Filozof 5 myśli...
Filozof 5 podniósł lewy widelec.
Filozof 5 podniósł prawy widelec.
Filozof 5 je. Licznik: 324
Filozof 5 odłożył oba widelce.
Filozof 5 myśli...
Filozof 5 podniósł lewy widelec.
Filozof 5 podniósł prawy widelec.
Filozof 5 je. Licznik: 325
Filozof 5 odłożył oba widelce.
Filozof 5 myśli...
Filozof 5 podniósł lewy widelec.
Filozof 5 podniósł prawy widelec.
Filozof 5 je. Licznik: 326
Filozof 5 odłożył oba widelce.
Filozof 5 myśli...
Filozof 5 podniósł lewy widelec.
Filozof 5 podniósł prawy widelec.
Filozof 5 je. Licznik: 327
Filozof 5 odłożył oba widelce.
Filozof 5 myśli...
Filozof 5 podniósł lewy widelec.
Filozof 5 podniósł prawy widelec.
Filozof 5 je. Licznik: 328
Filozof 5 odłożył oba widelce.
```

tw3 > src > main > java > org > example > Fil5mon.java > Widelec

Rysunek 1: Rozwiązanie z symetrycznymi filozofami

3 Rozwiązanie z widelcami podnoszonymi jednocześnie

3.1 Implementacja programu

```
class Widelec {
    private boolean zajety = false;

    public synchronized boolean podnies() {
        if (!zajety) {
            zajety = true;
            return true;
        }
        return false;
    }

    public synchronized void odloz() {
        zajety = false;
    }
}

class Filozof extends Thread {
    private int _licznik = 0;
    private final Widelec lewyWidelec;
    private final Widelec prawyWidelec;

    public Filozof(Widelec lewy, Widelec prawy) {
        this.lewyWidelec = lewy;
        this.prawyWidelec = prawy;
    }

    public void run() {
        try {
            while (true) {
                // Filozof myśli
                System.out.println(Thread.currentThread().getName() + " myśli...");
                Thread.sleep((int) (Math.random() * 100));

                // Próbuje podnieść oba widelce jednocześnie
                boolean obaWidelcePodniesione = false;

                while (!obaWidelcePodniesione) {
                    // Próbuje podnieść lewy i prawy widelec
                    synchronized (lewyWidelec) {
                        if (lewyWidelec.podnies()) {

```

```

        synchronized (prawyWidielec) {
            if (prawyWidielec.podnies()) {
                obaWidelcePodniesione = true;
            } else {
                lewyWidielec.odloz();
            }
        }
    }
    // Jeśli nie udało się podnieść obu, próbuje ponownie
    if (!obaWidelcePodniesione) {
        Thread.sleep((int) (Math.random() * 50));
    }
}

// Filozof je
_licznik++;
System.out.println(Thread.currentThread().getName() +
    " je. Licznik: " + _licznik);
Thread.sleep((int) (Math.random() * 100));

// Filozof odkłada oba widelce
lewyWidielec.odloz();
prawyWidielec.odloz();
System.out.println(Thread.currentThread().getName() +
    " odłożył oba widelce.");
}
} catch (InterruptedException e) {
    e.printStackTrace();
}
}
}

```

```

public class Fil5mon {
    public static void main(String[] args) {
        int liczbaFilozofow = 5;
        Widelec[] widelce = new Widelec[liczbaFilozofow];
        Filozof[] filozofowie = new Filozof[liczbaFilozofow];

        // Inicjalizacja widelców
        for (int i = 0; i < liczbaFilozofow; i++) {
            widelce[i] = new Widelec();
        }

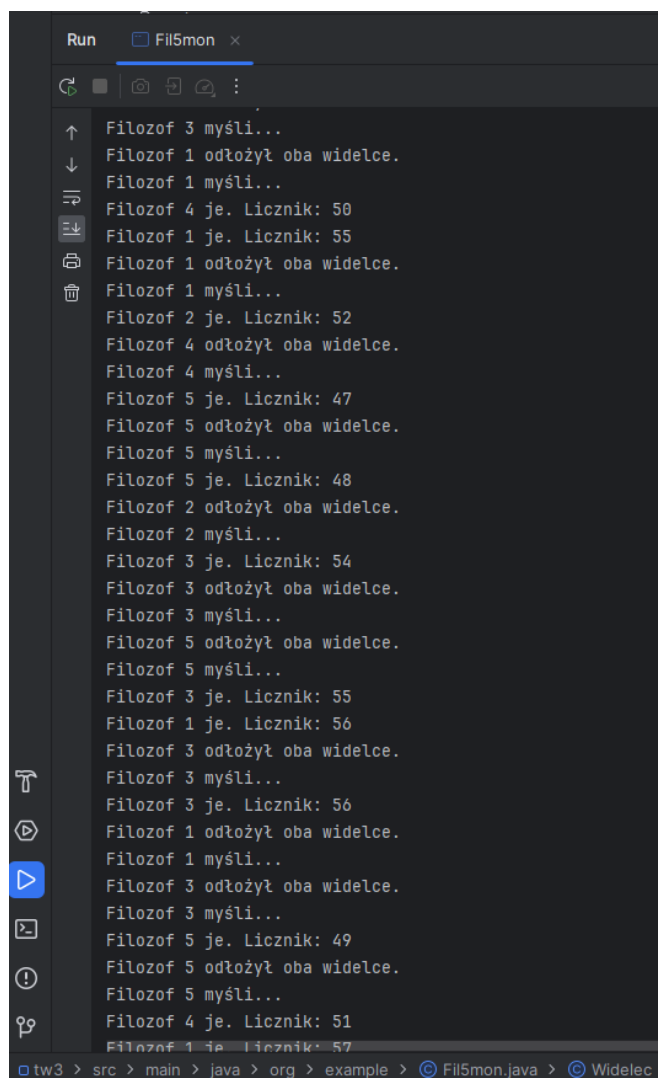
        // Inicjalizacja filozofów
        for (int i = 0; i < liczbaFilozofow; i++) {
            Widelec lewy = widelce[i];
            Widelec prawy = widelce[(i + 1) % liczbaFilozofow];
            filozofowie[i] = new Filozof(lewy, prawy);
            filozofowie[i].setName("Filozof " + (i + 1));
            filozofowie[i].start();
        }
    }
}

```

3.2 Jaki problem może wystąpić?

Główny problem, który może wystąpić w takiej sytuacji, to zagłodzenie (starvation), a niekoniecznie blokada (deadlock). Zagłodzenie może wystąpić, gdy:

- Filozof wielokrotnie nie udaje się zdobyć obu widelców naraz, podczas gdy inni filozofowie mają więcej szczęścia i jedzą częściej.
- W efekcie, niektórym filozofom może nigdy nie udać się zjeść, ponieważ inne wątki ciągle zdobywają widelce przed nimi.



```
Run Fil5mon x
Filozof 3 myśli...
Filozof 1 odłożył oba widelce.
Filozof 1 myśli...
Filozof 4 je. Licznik: 50
Filozof 1 je. Licznik: 55
Filozof 1 odłożył oba widelce.
Filozof 1 myśli...
Filozof 2 je. Licznik: 52
Filozof 4 odłożył oba widelce.
Filozof 4 myśli...
Filozof 5 je. Licznik: 47
Filozof 5 odłożył oba widelce.
Filozof 5 myśli...
Filozof 5 je. Licznik: 48
Filozof 2 odłożył oba widelce.
Filozof 2 myśli...
Filozof 3 je. Licznik: 54
Filozof 3 odłożył oba widelce.
Filozof 3 myśli...
Filozof 5 odłożył oba widelce.
Filozof 5 myśli...
Filozof 3 je. Licznik: 55
Filozof 1 je. Licznik: 56
Filozof 3 odłożył oba widelce.
Filozof 3 myśli...
Filozof 3 je. Licznik: 56
Filozof 1 odłożył oba widelce.
Filozof 1 myśli...
Filozof 3 odłożył oba widelce.
Filozof 3 myśli...
Filozof 5 je. Licznik: 49
Filozof 5 odłożył oba widelce.
Filozof 5 myśli...
Filozof 4 je. Licznik: 51
Filozof 1 je. Licznik: 57
```

Rysunek 2: Rozwiązanie z widelcami podnoszonymi jednocześnie

4 Rozwiązanie z lokajem

4.1 Implementacja programu

```
import java.util.concurrent.Semaphore;

class Widelec {
    private boolean zajety = false;

    public synchronized void podnies() {
        while (zajety) {
            try {
                wait(); // Czekaj, aż widelec będzie dostępny
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        zajety = true;
    }

    public synchronized void odloz() {
        zajety = false;
        notifyAll(); // Powiadamia inne wątki, że widelec jest dostępny
    }
}

class Filozof extends Thread {
    private int _licznik = 0;
    private final Widelec lewyWidelec;
    private final Widelec prawyWidelec;
    private final Semaphore lokaj;

    public Filozof(Widelec lewy, Widelec prawy, Semaphore lokaj) {
        this.lewyWidelec = lewy;
        this.prawyWidelec = prawy;
        this.lokaj = lokaj;
    }

    @Override
    public void run() {
        try {
            while (true) {
                // Filozof myśli
                System.out.println(Thread.currentThread().getName() + " myśli...");
                Thread.sleep((int) (Math.random() * 100));
            }
        }
    }
}
```

```

        // Lokaj pozwala filozofowi na jedzenie
        lokaj.acquire();

        // Filozof próbuje podnieść oba widelce
        lewyWidelec.podnies();
        System.out.println(Thread.currentThread().getName() +
            " podniósł lewy widelec.");

        prawyWidelec.podnies();
        System.out.println(Thread.currentThread().getName() +
            " podniósł prawy widelec.");

        // Filozof je
        _licznik++;
        System.out.println(Thread.currentThread().getName() +
            " je. Licznik: " + _licznik);
        Thread.sleep((int) (Math.random() * 100));

        // Filozof odkłada oba widelce
        lewyWidelec.odloz();
        prawyWidelec.odloz();
        System.out.println(Thread.currentThread().getName() +
            " odłożył oba widelce.");

        // Filozof opuszcza stół, zwalniając miejsce
        lokaj.release();
    }
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
}
}
}

```

```

public class Fil5mon {
    public static void main(String[] args) {
        int liczbaFilozofow = 5;
        Widelec[] widelce = new Widelec[liczbaFilozofow];
        Filozof[] filozofowie = new Filozof[liczbaFilozofow];

        // Inicjalizacja widelców
        for (int i = 0; i < liczbaFilozofow; i++) {
            widelce[i] = new Widelec();
        }

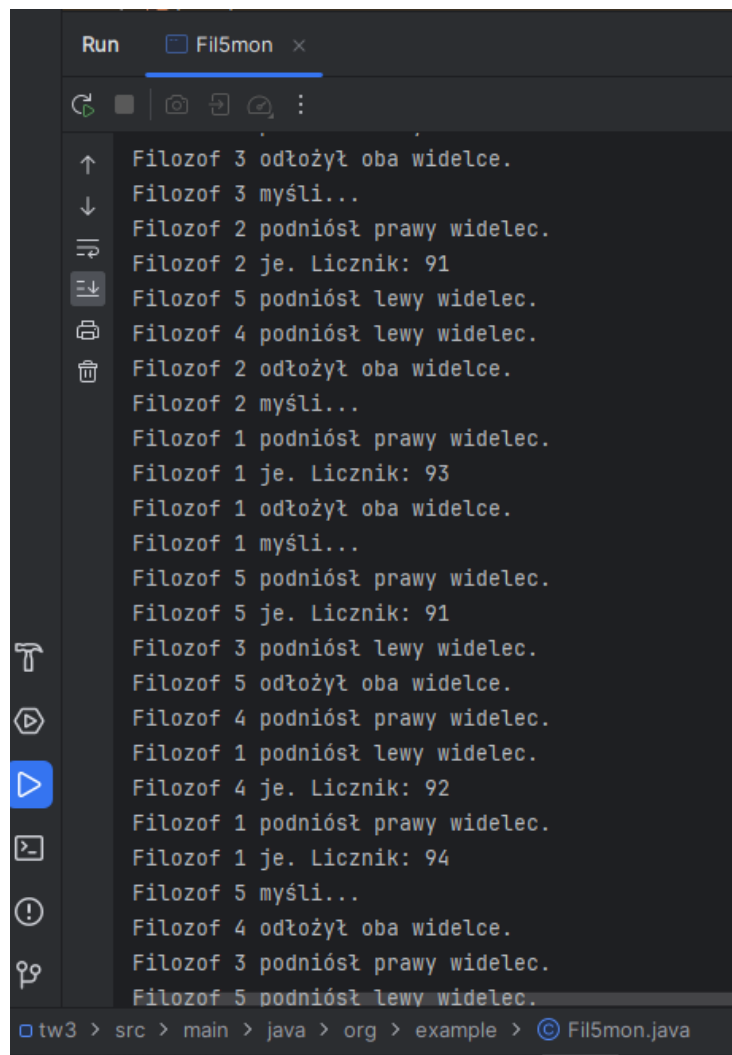
        Semaphore lokaj = new Semaphore(4);

        // Inicjalizacja filozofów
        for (int i = 0; i < liczbaFilozofow; i++) {
            Widelec lewy = widelce[i];
            Widelec prawy = widelce[(i + 1) % liczbaFilozofow];
            filozofowie[i] = new Filozof(lewy, prawy, lokaj);
            filozofowie[i].setName("Filozof " + (i + 1));
            filozofowie[i].start();
        }
    }
}

```

4.2 Dlaczego to rozwiązanie jest skutecznie?

- **Brak blokady:** Lokaj ogranicza liczbę filozofów próbujących jeść jednocześnie, dzięki czemu zawsze przynajmniej jeden filozof będzie miał dostęp do obu widelców.
- **Brak zagłodzenia:** Każdy filozof ma równą szansę na zdobycie zasobów, ponieważ lokaj zarządza dostępem do stołu.



```
Run Fil5mon x
Filozof 3 odłożył oba widelce.
Filozof 3 myśli...
Filozof 2 podniósł prawy widelec.
Filozof 2 je. Licznik: 91
Filozof 5 podniósł lewy widelec.
Filozof 4 podniósł lewy widelec.
Filozof 2 odłożył oba widelce.
Filozof 2 myśli...
Filozof 1 podniósł prawy widelec.
Filozof 1 je. Licznik: 93
Filozof 1 odłożył oba widelce.
Filozof 1 myśli...
Filozof 5 podniósł prawy widelec.
Filozof 5 je. Licznik: 91
Filozof 3 podniósł lewy widelec.
Filozof 5 odłożył oba widelce.
Filozof 4 podniósł prawy widelec.
Filozof 1 podniósł lewy widelec.
Filozof 4 je. Licznik: 92
Filozof 1 podniósł prawy widelec.
Filozof 1 je. Licznik: 94
Filozof 5 myśli...
Filozof 4 odłożył oba widelce.
Filozof 3 podniósł prawy widelec.
Filozof 5 podniósł lewy widelec.
tw3 > src > main > java > org > example > Fil5mon.java
```

Rysunek 3: Rozwiązanie z lokajem

5 Pomiary dla każdego rozwiązania

5.1 Implementacja programu do pomiarów

```
import java.util.concurrent.Semaphore;

class Widelec {
    private boolean zajety = false;

    public synchronized boolean podnies() {
        if (!zajety) {
            zajety = true;
            return true;
        }
        return false;
    }

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

class Filozof extends Thread {
    private int _licznik = 0;
    private final Widelec lewyWidelec;
    private final Widelec prawyWidelec;
    private final Semaphore lokaj;
    private final boolean useButler;

    public Filozof(Widelec lewy, Widelec prawy, Semaphore lokaj, boolean useButler) {
        this.lewyWidelec = lewy;
        this.prawyWidelec = prawy;
        this.lokaj = lokaj;
        this.useButler = useButler;
    }

    public int getLicznik() {
        return _licznik;
    }

    @Override
    public void run() {
        try {
            while (_licznik < 100) { // Każdy filozof wykonuje 100 cykli jedzenia
                // Filozof myśli
            }
        }
    }
}
```

```

        Thread.sleep((int) (Math.random() * 100));

        if (useButler) {
            lokaj.acquire();
        }

        boolean obaWidelcePodniesione = false;

        while (!obaWidelcePodniesione) {
            synchronized (lewyWidelec) {
                if (lewyWidelec.podnies()) {
                    synchronized (prawyWidelec) {
                        if (prawyWidelec.podnies()) {
                            obaWidelcePodniesione = true;
                        } else {
                            lewyWidelec.odloz();
                        }
                    }
                }
            }
            if (!obaWidelcePodniesione) {
                Thread.sleep((int) (Math.random() * 50));
            }
        }

        // Filozof je
        _licznik++;
        Thread.sleep((int) (Math.random() * 100));

        lewyWidelec.odloz();
        prawyWidelec.odloz();

        if (useButler) {
            lokaj.release();
        }
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
}

}

public class FilozofowieTest {
    public static void main(String[] args) {
        int liczbaFilozofow = 5;
        Widelec[] widelce = new Widelec[liczbaFilozofow];
    }
}

```

```

        Filozof[] filozofowie = new Filozof[liczbaFilozofow];

        // Inicjalizacja widelców
        for (int i = 0; i < liczbaFilozofow; i++) {
            widelce[i] = new Widelec();
        }

        testRozwiazanie("Symetryczne", widelce, filozofowie, false, null);
        testRozwiazanie("Jednoczesne", widelce, filozofowie, false, null);
        Semaphore lokaj = new Semaphore(4);
        testRozwiazanie("Lokaj", widelce, filozofowie, true, lokaj);
    }

    public static void testRozwiazanie(String nazwa, Widelec[] widelce,
        Filozof[] filozofowie, boolean useButler, Semaphore lokaj) {
        System.out.println("\n--- Test dla rozwiązania: " + nazwa + " ---");
        long startTime = System.currentTimeMillis();

        for (int i = 0; i < 5; i++) {
            Widelec lewy = widelce[i];
            Widelec prawy = widelce[(i + 1) % 5];
            filozofowie[i] = new Filozof(lewy, prawy, lokaj, useButler);
            filozofowie[i].setName("Filozof " + (i + 1));
            filozofowie[i].start();
        }

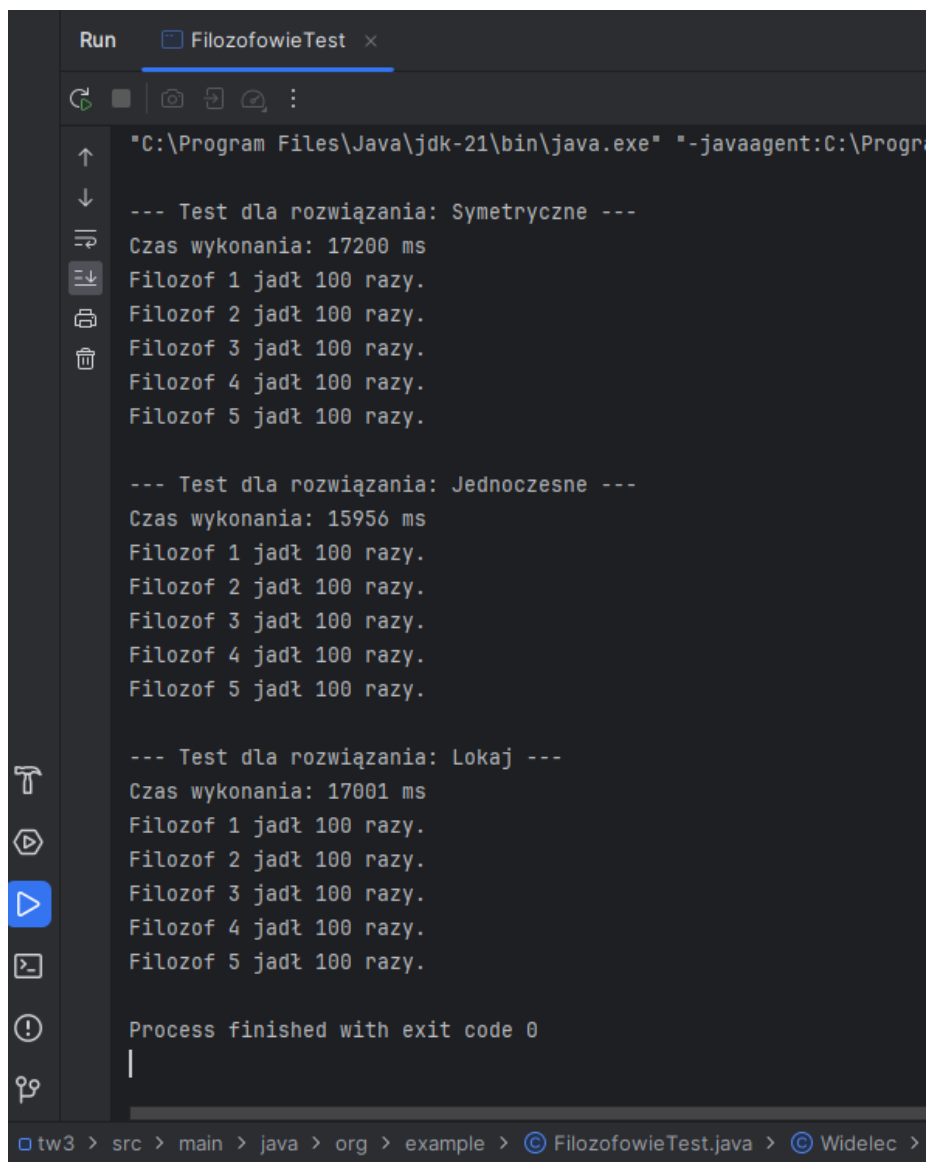
        for (Filozof filozof : filozofowie) {
            try {
                filozof.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        long endTime = System.currentTimeMillis();
        System.out.println("Czas wykonania: " +
            (endTime - startTime) + " ms");

        for (Filozof filozof : filozofowie) {
            System.out.println(filozof.getName() + " jadł " +
                filozof.getLicznik() + " razy.");
        }
    }
}

```

5.2 Wyniki programu



```
Run FilozofowieTest x
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Progr...

--- Test dla rozwiązania: Symetryczne ---
Czas wykonania: 17200 ms
Filozof 1 jadł 100 razy.
Filozof 2 jadł 100 razy.
Filozof 3 jadł 100 razy.
Filozof 4 jadł 100 razy.
Filozof 5 jadł 100 razy.

--- Test dla rozwiązania: Jednoczesne ---
Czas wykonania: 15956 ms
Filozof 1 jadł 100 razy.
Filozof 2 jadł 100 razy.
Filozof 3 jadł 100 razy.
Filozof 4 jadł 100 razy.
Filozof 5 jadł 100 razy.

--- Test dla rozwiązania: Lokaj ---
Czas wykonania: 17001 ms
Filozof 1 jadł 100 razy.
Filozof 2 jadł 100 razy.
Filozof 3 jadł 100 razy.
Filozof 4 jadł 100 razy.
Filozof 5 jadł 100 razy.

Process finished with exit code 0
```

Rysunek 4: Wyniki pomiarów

5.3 Wnioski

1. Wydajność:

- Rozwiązanie z jednoczesnym podnoszeniem widelców okazało się najwydajniejsze, ponieważ nie wprowadza dodatkowych mechanizmów synchronizacji poza blokadą na widelcach.
- Rozwiązanie z lokajem jest nieco wolniejsze ze względu na dodatkową synchronizację z użyciem semafora, który kontroluje dostęp do stołu.

2. Stabilność i bezpieczeństwo:

- Rozwiązanie z lokajem jest najbardziej bezpieczne, ponieważ gwarantuje brak blokady i równy dostęp do zasobów, kosztem niewielkiego spadku wydajności.
- Rozwiązania symetryczne i jednoczesne podnoszenie widelców również działają dobrze w tym konkretnym teście, ale w bardziej losowych sytuacjach mogą prowadzić do problemów takich jak blokada lub zgłódzenie, zwłaszcza gdy liczba filozofów lub widelców zostanie zmieniona.

3. Zastosowanie w praktyce:

- Jeśli zależy nam na prostocie i wydajności, rozwiązanie z jednoczesnym podnoszeniem widelców może być preferowane.
- Jeśli zależy nam na stabilności i braku ryzyka deadlocku, rozwiązanie z lokajem będzie lepszym wyborem, nawet kosztem nieco dłuższego czasu wykonania.

6 Bibliografia

Abraham Silberschatz, Peter B. Galvin, Greg Gagne. *Operating System Concepts*. Wiley, 2020. ISBN: 978-1119800361.

Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea. *Java Concurrency in Practice*. Addison-Wesley Professional, 2006. ISBN: 978-0321349606.

Andrew S. Tanenbaum, Herbert Bos. *Modern Operating Systems*. Pearson, 2015. ISBN: 978-0133591620.

Oracle. *The Java Tutorials - Concurrency*. Oracle, 2023.

Maurice Herlihy, Nir Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2020. ISBN: 978-0124159501.

A. S. Tanenbaum, M. Van Steen. *Distributed Systems: Principles and Paradigms*. Pearson, 2007. ISBN: 978-0132392273.