

**Zintegrowany Program Rozwoju  
Akademii Górniczo-Hutniczej w Krakowie**  
Nr umowy: POWR.03.05.00-00-Z307/17

**Instrukcja do ćwiczeń laboratoryjnych**

<b>Nazwa przedmiotu</b>	Teoria Współbieżności
<b>Numer ćwiczenia</b>	3
<b>Temat ćwiczenia</b>	Rozwiązanie problemu pięciu filozofów w różnych paradygmatach programowania współbieżnego

Poziom studiów	I stopień
Kierunek	Informatyka
Forma studiów	Stacjonarne
Semestr	5

dr hab. inż. Bartosz Baliś



Wydział Informatyki, Elektroniki i Telekomunikacji  
Kraków, 2019

## 1. Cel ćwiczenia

Celem ćwiczenia jest porównanie różnych rozwiązań problemu pięciu filozofów w dwóch paradygmatach programowania współbieżnego:

- Programowanie wielowątkowe (Java)
- Programowanie asynchroniczne (JavaScript/Node.js)

Ćwiczenie zakłada znajomość podstaw programowania współbieżnego w powyższych językach i środowiskach.

## 2. Wprowadzenie do ćwiczenia

**Problem pięciu filozofów** jest jednym z klasycznych problemów teorii współbieżności. Podstawowe sformułowanie problemu jest następujące :

- N filozofów zasiada przy okrągłym stole
- Pomiedzy sąsiednimi filozofami leży widelec (łącznie jest N widelców)
- Każdy filozof działa ciągle według schematu „myślenie - jedzenie - myślenie - jedzenie - ...”. Każdy z etapów (myślenie i jedzenie) jest skończony.
- Aby zjeść, filozof musi podnieść oba sąsiadujące widelce

**Zadanie:** zaprojektuj algorytm jednoczesnej alokacji współdzielonych zasobów (widelce) przez konkurujące procesy (filozofowie), tak aby uniknąć zakleszczenia i zagłodzenia.

Niektóre z rozwiązań problemu pięciu filozofów są następujące [1].

1. **Rozwiązanie naiwne (z możliwością blokady).** Każdy filozof czeka, aż wolny będzie lewy widelec, a następnie go podnosi (zajmuje), następnie podobnie postępuje z prawym widelcem.
2. **Rozwiązanie z możliwością zagłodzenia.** Każdy filozof sprawdza czy oba sąsiednie widelce są wolne i dopiero wtedy zajmuje je jednocześnie. Rozwiązanie to jest wolne od blokady, jednak w przypadku, gdy zawsze któryś z sąsiadów będzie zajęty jedzeniem, nastąpi zagłodzenie, gdyż oba widelce nigdy nie będą wolne.
3. **Rozwiązanie asymetryczne.** Filozofowie są ponumerowani. Filozof z parzystym numerem najpierw podnosi prawy widelec, filozof z nieparzystym numerem najpierw podnosi lewy widelec.
4. **Rozwiązanie z arbitrem.** Zewnętrzny arbiter (lokaj, kelner) pilnuje, aby jednocześnie co najwyżej czterech (w ogólnym przypadku N-1) filozofów konkurowało o widelce. Jeśli naraz wszyscy filozofowie będą chcieli jeść, arbiter powstrzymuje jednego z nich aż do czasu, gdy któryś z filozofów skończy jeść.

### 3. Plan ćwiczenia

#### 3.1. Implementacja - programowanie wielowątkowe

Zaimplementuj rozwiązanie problemu pięciu Filozofów w Javie, wykorzystując mechanizm semaforów, w następujących wariantach:

1. Wariant z możliwością zagłodzenia.
2. Wariant z arbitrem.

Można skorzystać z rozwiązań podanych w książce T. Weissa i Z. Gruzlewskiego [1] na stronach 55-56.

#### 3.2. Implementacja - programowanie asynchroniczne

W paradygmacie programowania asynchronicznego nie ma mechanizmów synchronizacji. Problem dostępu do współdzielonych zasobów można zatem oprzeć o algorytm BEB (Binary Exponential Backoff), podobny do stosowanego w protokole CSMA/CD, który jest wykorzystywany w sieci Ethernet do rozwiązania problemu jednoczesnego dostępu do wspólnego medium.

Korzystając z zadanego szkieletu programu w Node.js do poprawy tutaj:

1. Dokończ implementację funkcji podnoszenia widelca (Fork.acquire) w oparciu o algorytm BEB.
2. Zaimplementuj "naiwny" algorytm (filozof podnosi najpierw lewy, potem prawy widelec).
3. Zaimplementuj rozwiązanie asymetryczne: filozofowie z nieparzystym numerem najpierw podnoszą widelec lewy, z parzystym - prawy.
4. Zaimplementuj rozwiązanie z arbitrem (kelnerem).
5. Zaimplementuj rozwiązanie z jednoczesnym podnoszeniem widelców: filozof albo podnosi jednocześnie oba widelce, albo żadnego.

#### 3.3. Porównanie rozwiązań

Korzystając z wykonanych implementacji:

- Uruchom eksperymenty dla różnej liczby filozofów i dla każdego wariantu implementacji (nie powodującego zakleszczenia).
- **Zmierz średni czas oczekiwania każdego filozofa na dostęp do widelców.** Wykonaj kilka pomiarów dla każdego przypadku testowego.
- Wyniki przedstaw na wykresach porównawczych, dbając o odpowiednią wizualizację (można wykorzystać np. wykresy pudełkowe).
- Sformułuj i zapisz wnioski. Czy średnie czasy oczekiwania są wyższe dla wariantu z możliwością zagłodzenia? Czy brak mechanizmów synchronizacji zwiększa czas oczekiwania na dostęp do zasobów? (Porównanie Node.js vs. Java może to być trudne do oceny z uwagi na różne środowiska wykonawcze).

## 4. Sposób oceny

Ocena będzie oparta o następujące kryteria:

- Kompletność i poprawność implementacji rozwiązań: 60% (programowanie wielowątkowe 30%, programowanie asynchroniczne 30%)
- Wykonanie eksperymentów, opracowanie wyników i redakcja sprawozdania: 40%

## Literatura

- [1] Z. Weiss, T. Gruzlewski, Programowanie współbieżne i rozproszone, WNT, Warszawa 1993.
- [2] Problem uczujących filozofów:  
[https://pl.wikipedia.org/wiki/Problem\\_ucztujacych\\_filozofow](https://pl.wikipedia.org/wiki/Problem_ucztujacych_filozofow)
- [3] Algorytm Binary Exponential Backoff:  
[http://pl.wikipedia.org/wiki/Binary\\_Exponential\\_Backoff](http://pl.wikipedia.org/wiki/Binary_Exponential_Backoff)