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

Instrukcja do ćwiczeń laboratoryjnych

Nazwa przedmiotu	**Teoria współbieżności**
Numer ćwiczenia	
Temat ćwiczenia	**Rozwiązanie problemu pięciu filozofów w różnych paradygmatach programowania współbieżnego**

Poziom studiów	I stopień
Kierunek	Informatyka
Forma i tryb studiów	Stacjonarne
Semestr	5

Bartosz Baliś
autor instrukcji

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[1]:

- N filozofów zasiada przy okrągłym stole
- Pomiędzy 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[2].

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. F**ilozofowie 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/ program ćwiczenia

3.1. Implementacja - programowanie wielowątkowe

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

1. Wariant z możliwością zagłodzenia.

[1] Por. Z. Weiss, T. Gruźlewski, Programowanie współbieżne i rozproszone, Warszawa 1993, ss. 29-30. Również https://pl.wikipedia.org/wiki/Problem_ucztuj%C4%85cych_filozof%C3%B3w

[2] Z. Weiss, Gruźlewski, ibid., ss. 30-31

2. Wariant z arbitrem.

Można skorzystać z rozwiązań podanych w książce T. Weissa i Z. Gruźlewskiego.[3]

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)[4], 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[5]:

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 <u>każdego filozofa</u> 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).

2. Sposób oceny/ uzyskania zaliczenia

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%

3. Literatura

1. Z. Weiss, T. Gruźlewski, Programowanie współbieżne i rozproszone, WNT, Warszawa 1993.
2. Problem ucztujących filozofów: https://pl.wikipedia.org/wiki/Problem_ucztuj%C4%85cych_filozof%C3%B3w
3. Algorytm Binary Exponential Backoff: http://pl.wikipedia.org/wiki/Binary_Exponential_Backoff

Załączniki

N/D

[3] Z. Weiss, T. Gruźlewski, Ibid., ss. 55-56.
[4] Algorytm Binary Exponential Backoff: http://pl.wikipedia.org/wiki/Binary_Exponential_Backoff
[5] Szkielet programu można pobrać z: https://github.com/balis/conc-phil5