Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Systemy Operacyjne

Problem Ucztujących Filozofów

Dining Philosophers Problem

Implementacja i analiza porównawcza trzech strategii synchronizacji wielowątkowej

Autor:
Yaroslav Perepilka

Prowadzący:
dr inż. Mariusz Makuchowski

Wrocław, 6 stycznia 2025

Contents

1 Wstęp

Problem ucztujących filozofów (ang. *Dining Philosophers Problem*) jest klasycznym problemem synchronizacji procesów, sformułowanym przez Edsgara Dijkstrę w 1965 roku. Problem ten ilustruje wyzwania związane z zapobieganiem zakleszczeniom (*deadlock*) i zagłodzeniu (*starvation*) w systemach wielowątkowych.

1.1 Opis problemu

Pięciu filozofów siedzi przy okrągłym stole. Każdy filozof na przemian myśli i je. Do jedzenia potrzebne są dwa widelce – lewy i prawy. Problem polega na zaprojektowaniu takiego protokołu, który:

- Zapobiega zakleszczeniom (deadlock)
- Zapobiega zagłodzeniu pojedynczych filozofów (starvation)
- Maksymalizuje współbieżność (concurrency)
- Zapewnia sprawiedliwy dostęp do zasobów (fairness)

1.2 Cel projektu

Celem niniejszego projektu jest:

1. Implementacja trzech różnych strategii synchronizacji
2. Analiza porównawcza wydajności i sprawiedliwości
3. Badanie wpływu liczby filozofów na zachowanie systemu
4. Wykrycie i demonstracja sytuacji zakleszczenia

2 Implementacja

2.1 Środowisko techniczne

Projekt został zrealizowany w języku Python 3 z wykorzystaniem biblioteki standardowej `threading`. Wybór języka Python pozwala na czytelną implementację oraz łatwą analizę wyników.

2.1.1 Specyfikacja sprzętowa

Testy wydajnościowe przeprowadzono na następującym sprzęcie:

- **Procesor:** AMD Ryzen 7 6800H with Radeon Graphics
- **Liczba rdzeni fizycznych:** 8
- **Liczba wątków (z SMT):** 16
- **System operacyjny:** Linux

2.1.2 Wpływ wielordzeniowości na problem

Liczba dostępnych rdzeni procesora ma kluczowe znaczenie dla problemu ucztujących filozofów:

1. **Prawdziwa współbieżność:** Przy 16 dostępnych wątkach sprzętowych możliwa jest rzeczywista równoległa praca wielu filozofów jednocześnie (w przeciwieństwie do pseudo-równoległości na jednordzeniowych systemach).

2. **Intensyfikacja race conditions:** Większa liczba rdzeni zwiększa prawdopodobieństwo wystąpienia rzeczywistych wyścigów (race conditions) i zakleszczenia, co lepiej pokazuje problemy synchronizacji.

3. **Skalowalność:** Przy testach z 32 filozofami na 16 wątkach sprzętowych widoczne staje się znaczenie efektywnej synchronizacji – w danym momencie jedynie połowa filozofów może wykonywać się równolegle.

4. **Overhead context switching:** Gdy liczba filozofów przekracza liczbę wątków sprzętowych, scheduler systemu operacyjnego musi przełączać konteksty, co wpływa na wydajność.

W przypadku strategii **deadlock**, wielordzeniowość przyspiesza wystąpienie zakleszczenia – wszystkie wątki mogą niemal jednocześnie zająć lewe widelce, prowadząc do natychmiastowego deadlocka.

Dla strategii **hierarchy** i **asymmetric**, większa liczba rdzeni pozwala na wyższy throughput (więcej posiłków na sekundę), gdyż więcej filozofów może jeść równolegle bez blokowania się nawzajem.

2.2 Zaimplementowane strategie

2.2.1 Strategia 1: Deadlock (Podejście naiwne)

Wszyscy filozofowie działają jednolicie – najpierw podnoszą lewy widelec, następnie prawy. Ta strategia prowadzi do zakleszczenia, gdy wszyscy filozofowie jednocześnie podniosą lewy widelec i czekają na prawy.

2.2.2 Strategia 2: Hierarchy (Rozwiązanie Dijkstry)

Widelce numerowane są od 0 do N-1. Każdy filozof zawsze podnosi widelec o niższym numerze jako pierwszy. To rozwiązanie przerywa cykliczne oczekiwanie (*circular wait*), eliminując możliwość zakleszczenia.

2.2.3 Strategia 3: Asymmetric (Filozofowie parzyści/nieparzyści)

Filozofowie o parzystych indeksach podnoszą najpierw lewy widelec, a o nieparzystych – prawy. Przerwanie symetrii działań zapobiega jednoczesnej próbie zajęcia tych samych zasobów.

3 Metodologia badań

3.1 Parametry testów

Przeprowadzono testy dla następujących konfiguracji:

- **Liczba filozofów:** 3, 5, 16, 32
- **Czas symulacji:** 30s, 60s, 180s (3 min), 600s (10 min), 1200s (20 min)
- **Strategie:** deadlock, hierarchy, asymmetric

3.2 Mierzone metryki

- Całkowita liczba posiłków (*total meals*)
- Średnia liczba posiłków na filozofa
- Przepustowość systemu (posiłki/sekundę)
- Współczynnik sprawiedliwości (*fairness coefficient*)
- Odchylenie standardowe liczby posiłków
- Wykrycie zakleszczenia

4 Wyniki

[Tutaj umieścić wykresy, tabele i analizę wyników z pliku performance_ analysis.csv]

5 Wnioski

[Podsumowanie wyników i wnioski końcowe]

6 Bibliografia

1. Dijkstra, E. W. (1971). *Hierarchical ordering of sequential processes.* Acta Informatica.
2. Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Pearson.
3. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.