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

4.1 Strategia Deadlock – demonstracja zakleszczenia

Strategia deadlock, jak oczekiwano, prowadzi do całkowitego zakleszczenia systemu. Wszystkie testy zakończyły się deadlockiem – żaden filozof nie zdołał zjeść ani jednego posiłku.

Table 1: Wyniki strategii Deadlock

Liczba filozofów	Czas [s]	Posiłki	Zakleszczeni
3	1200	0	3
5	1200	0	5
16	1200	0	16
32	1200	0	32

Obserwacje:

- Deadlock występuje niemal natychmiast (w pierwszych sekundach symulacji)
- Wszyscy filozofowie zostają zablokowany z jednym widelcem w ręku
- Niezależnie od liczby filozofów, wynik jest identyczny – zero posiłków
- System nie może wyjść z deadlocka samoistnie

Table 2: Wyniki strategii Hierarchy – pełne wyniki

Filozofowie	Czas [s]	Posiłki	Throughput	Fairness [%]
3	30	152	4.97	66.15
3	60	301	4.98	66.67
3	180	900	4.99	64.38
3	600	2996	4.99	65.28
3	1200	5990	4.99	67.36
5	30	303	9.90	80.30
5	60	601	9.93	76.69
5	180	1799	9.96	81.96
5	600	5987	9.97	78.69
5	1200	11969	9.97	77.70
16	30	1200	39.54	100.00
16	60	2243	37.22	94.44
16	180	7106	39.40	98.44
16	600	23755	39.56	99.53
16	1200	47563	39.62	99.80
32	30	2162	71.11	81.33
32	60	4548	75.43	92.62
32	180	14048	77.87	96.42
32	600	44849	74.68	98.72
32	1200	94816	78.97	100.00

4.2 Strategia Hierarchy – rozwiązanie Dijkstry

Strategia hierarchy całkowicie eliminuje możliwość deadlocka poprzez wprowadzenie hierarchii zasobów. Wszyscy filozofowie zawsze pobierają widelec o niższym numerze jako pierwszy.

Kluczowe obserwacje:

- **Zero deadlocków** – strategia skutecznie zapobiega zakleszczeniom
- **Skalowalność:** Przepustowość rośnie liniowo z liczbą filozofów
- **Stabilność:** Throughput pozostaje stały niezależnie od czasu symulacji
- **Fairness coefficient:** Rośnie z czasem symulacji, osiągając 100% dla 32 filozofów w 1200s
- Najlepsza sprawiedliwość przy większej liczbie filozofów

4.3 Strategia Asymmetric – przerwanie symetrii

Strategia asymmetryczna (parzyści filozofowie: lewy→prawy, nieparzyści: prawy→lewy) również skutecznie zapobiega deadlockom.

Kluczowe obserwacje:

- **Zero deadlocków** – równie skuteczna jak hierarchy
- **Wyższa przepustowość:** Przy 32 filozofach osiąga 79.45 meals/s (vs 78.97 hierarchy)
- **Doskonała sprawiedliwość:** Fairness coefficient 98-100% dla większych grup

Table 3: Wyniki strategii Asymmetric – pełne wyniki

Filozofowie	Czas [s]	Posiłki	Throughput	Fairness [%]
3	30	152	4.97	61.76
3	60	301	4.98	65.38
3	180	900	4.98	62.66
3	600	2996	4.99	64.93
3	1200	5990	4.99	65.28
5	30	302	9.87	87.88
5	60	601	9.93	81.16
5	180	1797	9.96	80.15
5	600	5987	9.97	82.24
5	1200	11873	9.89	88.88
16	30	1193	39.36	98.67
16	60	2370	39.30	98.66
16	180	7134	39.57	99.55
16	600	23832	39.70	99.93
16	1200	47702	39.73	99.93
32	30	2399	79.09	98.67
32	60	4769	78.95	99.33
32	180	14286	79.20	99.55
32	600	47649	79.36	99.93
32	1200	95409	79.45	99.97

- **Bardzo niska wariancja:** Standard deviation < 1 dla 16 i 32 filozofów
- Lepsze wykorzystanie zasobów dzięki asymetrycznemu dostępowi

4.4 Porównanie strategii

Table 4: Porównanie strategii dla 32 filozofów, 1200s

Metryka	Deadlock	Hierarchy	Asymmetric
Całkowite posiłki	0	94816	95409
Throughput [meals/s]	0	78.97	79.45
Deadlock detected	TAK	NIE	NIE
Fairness coefficient	-	100%	99.97%
Std deviation	0	0.0	0.5

4.5 Wpływ liczby filozofów na wydajność

Dla strategii hierarchy i asymmetric:

1. **3 filozofy:** ≈ 5 meals/s
   - Ograniczona równoległość (max 1 para jedząca jednocześnie)
   - Wysoka sprawiedliwość utrudniona (66-67%)
2. **5 filozofów:** ≈ 10 meals/s (2x wzrost)

- Max 2 pary mogą jeść jednocześnie
- Lepsza sprawiedliwość (77-89%)

3. **16 filozofów:** $\approx$ 40 meals/s (4x wzrost)
   - Wysokie wykorzystanie 16 wątków sprzętowych
   - Doskonała sprawiedliwość (98-100%)
4. **32 filozofy:** $\approx$ 79 meals/s (8x wzrost)
   - Przekroczenie liczby wątków sprzętowych ($32 > 16$)
   - Overhead context switching, ale wciąż wysoka wydajność
   - Najlepsza sprawiedliwość w długich testach

4.6 Analiza sprawiedliwości (Fairness)

Współczynnik sprawiedliwości = (min_meals / max_meals) $\times$ 100%

- **Hierarchy:** 64-100%, rośnie z liczbą filozofów i czasem symulacji
- **Asymmetric:** 61-100%, również rośnie z parametrami
- W testach 1200s obie strategie osiągają $\geq$ 99% sprawiedliwości
- Dla 32 filozofów w długim teście: niemal idealna równość posiłków

5 Wnioski

5.1 Podsumowanie wyników

Przeprowadzone eksperymenty potwierdzają teoretyczne przewidywania dotyczące problemu ucztujących filozofów:

1. **Deadlock jest realnym zagrożeniem:** Naiwne podejście prowadzi do natychmiastowego zakleszczenia niezależnie od liczby filozofów czy czasu symulacji.
2. **Hierarchia zasobów działa:** Rozwiązanie Dijkstry całkowicie eliminuje deadlock, zapewniając stabilną wydajność i wysoką sprawiedliwość.
3. **Asymetria jest efektywna:** Przerwanie symetrii działań filozofów nie tylko zapobiega deadlockowi, ale również oferuje nieco wyższą przepustowość niż hierarchy.
4. **Wielordzeniowość ma znaczenie:** Na procesorze 8-rdzeniowym (16 wątków) widoczny jest liniowy wzrost wydajności do 16 filozofów. Przy 32 filozofach pojawia się overhead context switching, ale system pozostaje efektywny.

5.2 Porównanie strategii

Hierarchy (Dijkstra):

+ Matematycznie dowiedlna poprawność
+ Deterministyczna kolejność dostępu do zasobów
+ Doskonała sprawiedliwość w długich symulacjach (100%)
– Nieco niższa przepustowość niż asymmetric przy dużej liczbie filozofów

Asymmetric (Parzyści/Nieparzyści):

+ Najwyższa przepustowość (79.45 meals/s dla 32 filozofów)
+ Prostsza implementacja niż hierarchy
+ Bardzo dobra sprawiedliwość (99.97%)
+ Niższa wariancja w rozdziale posiłków
– Mniej intuicyjna niż hierarchy dla małych grup

5.3 Wnioski końcowe

1. Wybór strategii synchronizacji ma **krytyczne znaczenie** dla wydajności i bezpieczeństwa systemu wielowątkowego.
2. W systemach produkcyjnych należy **zawsze** stosować sprawdzone mechanizmy zapobiegania deadlockom (hierarchy, asymmetry, timeouts).
3. Przy projektowaniu systemów współbieżnych konieczne jest uwzględnienie:
    - Liczby dostępnych wątków sprzętowych
    - Oczekiwanego poziomu równoległości
    - Wymagań co do sprawiedliwości dostępu
4. **Testing w warunkach realnych** (prawdziwa wielordzeniowość) ujawnia problemy niewidoczne w symulacjach jednordzeniowych.
5. Dla systemów wymagających najwyższej wydajności, strategia **asymmetric** oferuje najlepsze wyniki przy zachowaniu bezpieczeństwa.

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.