Problem biesiadujących filozofów

Jan Góra

17 stycznia 2018

Treść zadania

Problem oraz treść zadania

Przy okrągłym stole biesiaduje N filozofów i każdy wykonuje jedną z dwóch czynności – albo je, albo rozmyśla. Przed każdym z nich znajduje się talerz, a pomiędzy każdą sąsiadującą parą filozofów leży pałeczka, a więc każdy ma przy sobie dwie sztuki – po swojej lewej i prawej stronie. Ponieważ jedzenie potrawy jest trudne przy użyciu jednej pałeczki, zakłada się, że każdy filozof korzysta z dwóch. Dodatkowo nie ma możliwości skorzystania z pałeczki, która nie znajduje się bezpośrednio przed daną osobą. Liczba dostępnych dań ograniczona jest przez liczbę M.

Uogólnić problem na problem synchronizacji procesów systemowych.

Podczas rozwiązywania zadania występić mogą następujące problemy:

- Każdy z nich zabierze lewą pałeczkę i będzie czekał na prawą (lub na odwrót) - **zakleszczenie**
- Niezależnie od zakleszczenia, dojść może do bardzo zróżnicowanego dostępu do posiłków - **zagłodzenie**

Rozwiązanie

Powyższy problem bardzo łatwo uogólnić na problem synchronizacji procesów systemowych. W rozwiązaniu, poprzez biesiadującego filozofa będziemy rozumieć proces, a dostęp do pałeczek oraz dostępnych dań będziemy rozumieć jako sekcję krytyczną, tzn. fragment programu odpowiadający za korzystanie z zasobów dzielonych. Do zarządzenia wyżej wymienionej sekcją użyte zostały semafory.

Jednym z rozwiązań problemu ucztujących filozów, a dokładniej problemu **zakleszczenia**, jest ograniczenie liczby dostępnych talerzy, przy warunku, że filozof bez talerza nie może podnieść pałeczki ze stołu. Łatwo zauważyć, że przy ograniczeniu talerzy poprzez liczbę $N - 1$, przynajmniej jeden z filozofów podniesie obie pałeczki, w następstwie czego przynajmniej jeden filozof zje w tym czasie jedną potrawę. Na mocy indukcji, z obserwacji tej wprost wynika, że wszystkie posiłki zostaną zjedzone oraz, że nie nastąpi zakleszczenie cykilczne, ponieważ aby owe zakleszczenie, przy tak sformułowanym zadaniu wystąpiło, wszystkie procesy muszą się wzajemnie „zakleszczyć".

Obok talerzy i pałeczek, kolejnym zasobem dzielonym jest liczba dostępnych dań, w wyniku czego podniesienie obu pałeczek nie jest warunkiem wystarczającym do zjedzenia posiłku – dany proces musi jeszcze wygrać „wyścig" po potrawę, co w teorii może nigdy nie nastąpić.
Rozwiązanie problemu **zagłodzenia** jest bardziej problematyczne. W praktyce niekoniecznie będziemy chcieli, aby każdy z procesów otrzymał równy dostęp do dzielonych zasobów oraz, aby każdy

z procesów odpytywał o zasób z tą samą częstotliwością. Jako, że filozof oprócz jedzenia również rozmyśla, rozmyślanie blokuje go przed zjedzeniem posiłku na pewien czas. Zauważmy, że im szerszy przedział długości oczekiwania względem liczby wszystkich filozofów, tym dostęp do dzielonych zasobów będzie łatwiejszy, w wyniku czego dostęp do pałeczek będzie niemal równoznaczny ze zjedzeniem posiłku, a dostęp do talerza skutkować będzie dostępem do obu pałeczek. W dołączonym rozwiązaniu czas oczekiwania, w mikrosekundach, wybierany jest losowo z przedziału $[0, N)$, co w znaczeniu przenośnym oznacza: „wyliczanie aż do znudzenia kolejnych filozofów". Idea pomysłu jest taka, aby nie pozbywać się zupełnie rywalizacji o zasoby, ale jednocześnie rywalizację tę znacząco zmniejszyć.

Rozwiązanie umieszczone jest w katalogu *Filozofowie* dołączonym do sprawozdania.
Pałeczkom odpowiada tablica semaforów (**stick**[N]) o buforach rozmiaru 1.
Talerzom odpowiada semafor (**plates**) o buforze rozmiaru $N - 1$.
Pozostałej liczbie dostępnych posiłków odpoawiada zmienna typu *int* (**dishes_left**) oraz semafor o buforze (**eating**) rozmiaru 1 kontrolującym dostęp do tej zmiennej.
Każdy proces (filozof) jest ponumerowany od 0 do $N - 1$, a swoje działanie kończy w przypadku gdy zmienna **dishes_left** jest równa 0.
Program kończy swoje działanie, kiedy nie ma żadnych aktywnych procesów pochodnych.
Przykładowy wynik uruchomienia programu, który możemy zaobserwować w grafice załączonej poniżej, działa zgodnie z oczekiwaniami, tj. procesy o najdłuższym okresie „rozmyślania" zużyły najmniej zasobów.

```
janek@janek-z570:~/Studia/SO/Filozofowie$ ./phi_race
[N: 10, M: 30, turn_on_plates: 1, turn_on_thinking: 1]
Philosoph No 5 has eaten 3 dishes with summary thinking of 0.001 ms
Philosoph No 3 has eaten 3 dishes with summary thinking of 0.001 ms
Philosoph No 1 has eaten 4 dishes with summary thinking of 0.001 ms
Philosoph No 0 has eaten 4 dishes with summary thinking of 0.001 ms
Philosoph No 8 has eaten 3 dishes with summary thinking of 0.001 ms
Philosoph No 2 has eaten 3 dishes with summary thinking of 0.001 ms
Philosoph No 6 has eaten 2 dishes with summary thinking of 0.003 ms
Philosoph No 4 has eaten 2 dishes with summary thinking of 0.003 ms
Philosoph No 7 has eaten 4 dishes with summary thinking of 0.001 ms
Philosoph No 9 has eaten 2 dishes with summary thinking of 0.003 ms
```

Testy

Program testowany był w przeróżnych konfiguracjach. Udawało mi się osiągnąć efekt zakleszczenia przy danych **N = 10**, **M = 10000000**, **turn_plates_on = false** oraz **turn_thinking_on = false**. Jednak, co jest bardzo interesujące, po zrestartowaniu komputera nie udaje mi się odtworzyć tego efektu. W przeprowadzonych testach, zgodnie z oczekiwaniami, liczba dań zjedzonych przez filozofów za każdym razem sumowała się do liczby M, co dowodzi tego, że brak jest wycieków dzielonych zasobów – w danym czasie dostęp do nich miał maksymalnie jeden proces. Dla kontrastu bez używania semafora **eating** liczba ta prawie nigdy sumowała się prawidłowo.

Przy ustawionych zmiennych: **turn_plates_on = false** i **turn_thinking_on = false**, przy zbliżonym stosunku M do N podział zasobów jest bardzo różny, a wraz ze wzrostem ilości dostępnych zasobów, podział ten stawał się bardziej równomierny.

```
janek@janek-z570:~/Studia/SO/Filozofowie$ ./phi_race 10 100 0 0
[N: 10, M: 100, turn_on_plates: 0, turn_on_thinking: 0]
Philosoph No 5 has eaten 40 dishes
Philosoph No 8 has eaten 0 dishes
Philosoph No 4 has eaten 0 dishes
Philosoph No 3 has eaten 45 dishes
Philosoph No 6 has eaten 0 dishes
Philosoph No 9 has eaten 1 dishes
Philosoph No 0 has eaten 12 dishes
Philosoph No 2 has eaten 0 dishes
Philosoph No 1 has eaten 2 dishes
Philosoph No 7 has eaten 0 dishes
```

```
janek@janek-z570:~/Studia/SO/Filozofowie$ ./phi_race 10 100 0 0
[N: 10, M: 100, turn_on_plates: 0, turn_on_thinking: 0]
Philosoph No 6 has eaten 3 dishes
Philosoph No 3 has eaten 22 dishes
Philosoph No 8 has eaten 14 dishes
Philosoph No 4 has eaten 15 dishes
Philosoph No 5 has eaten 3 dishes
Philosoph No 0 has eaten 21 dishes
Philosoph No 1 has eaten 13 dishes
Philosoph No 2 has eaten 7 dishes
Philosoph No 7 has eaten 0 dishes
Philosoph No 9 has eaten 2 dishes
```

```
janek@janek-z570:~/Studia/SO/Filozofowie$ ./phi_race 10 1000000 0 0
[N: 10, M: 1000000, turn_on_plates: 0, turn_on_thinking: 0]
Philosoph No 7 has eaten 101224 dishes
Philosoph No 4 has eaten 99563 dishes
Philosoph No 6 has eaten 100313 dishes
Philosoph No 3 has eaten 98339 dishes
Philosoph No 5 has eaten 99152 dishes
Philosoph No 9 has eaten 96473 dishes
Philosoph No 1 has eaten 104322 dishes
Philosoph No 2 has eaten 100579 dishes
Philosoph No 0 has eaten 98934 dishes
Philosoph No 8 has eaten 101101 dishes
```

Włączenie semafora przydzielającego talerze przyczyniło się do prawie całkowitego wyelimowania całkowitego zagłodzenia, jednak nie poprawiło to równomiernego przydziału zasobów. Natomiast wprowadzenie oczekiwania o losowej długości (w mikrosekundach, z przedziału $[0, N)$) poskutkowało niemal idealnie równomiernym podziałem zasobów. Dzieje się tak, ponieważ rywalizacja o zasób jedzenia jest znacząco mniejsza, co skutkuje mniejszą losowością wyników. W poprzednich przykładach, o dostęp do tego semafora mogło walczyć nawet $\frac{1}{2}N$ procesów. Najlepsze rezultaty uzyskuje się przy jednoczesnym aktywowaniu obu semaforów.

```
janek@janek-z570:~/Studia/SO/Filozofowie$ ./phi_race 10 100 1 0
[N: 10, M: 100, turn_on_plates: 1, turn_on_thinking: 0]
Philosoph No 2 has eaten 2 dishes
Philosoph No 6 has eaten 2 dishes
Philosoph No 7 has eaten 3 dishes
Philosoph No 3 has eaten 38 dishes
Philosoph No 8 has eaten 0 dishes
Philosoph No 1 has eaten 20 dishes
Philosoph No 0 has eaten 14 dishes
Philosoph No 4 has eaten 5 dishes
Philosoph No 9 has eaten 13 dishes
Philosoph No 5 has eaten 3 dishes
```

```
janek@janek-z570:~/Studia/SO/Filozofowie$ ./phi_race 10 100 0 1
[N: 10, M: 100, turn_on_plates: 0, turn_on_thinking: 1]
Philosoph No 4 has eaten 9 dishes with summary thinking of 0.000 ms
Philosoph No 2 has eaten 11 dishes with summary thinking of 0.009 ms
Philosoph No 3 has eaten 11 dishes with summary thinking of 0.009 ms
Philosoph No 1 has eaten 12 dishes with summary thinking of 0.008 ms
Philosoph No 0 has eaten 11 dishes with summary thinking of 0.009 ms
Philosoph No 6 has eaten 11 dishes with summary thinking of 0.009 ms
Philosoph No 9 has eaten 6 dishes with summary thinking of 0.003 ms
Philosoph No 5 has eaten 10 dishes with summary thinking of 0.007 ms
Philosoph No 8 has eaten 7 dishes with summary thinking of 0.001 ms
Philosoph No 7 has eaten 12 dishes with summary thinking of 0.008 ms
```

```
janek@janek-z570:~/Studia/SO/Filozofowie$ ./phi_race 10 100 1 1
[N: 10, M: 100, turn_on_plates: 1, turn_on_thinking: 1]
Philosoph No 2 has eaten 11 dishes with summary thinking of 0.006 ms
Philosoph No 6 has eaten 9 dishes with summary thinking of 0.008 ms
Philosoph No 0 has eaten 11 dishes with summary thinking of 0.006 ms
Philosoph No 9 has eaten 9 dishes with summary thinking of 0.008 ms
Philosoph No 3 has eaten 10 dishes with summary thinking of 0.008 ms
Philosoph No 4 has eaten 10 dishes with summary thinking of 0.008 ms
Philosoph No 7 has eaten 11 dishes with summary thinking of 0.006 ms
Philosoph No 8 has eaten 9 dishes with summary thinking of 0.008 ms
Philosoph No 5 has eaten 9 dishes with summary thinking of 0.008 ms
Philosoph No 1 has eaten 11 dishes with summary thinking of 0.006 ms
```

Podczas gdy $N = 10^3$ (liczba procesów) i $M = 10^7$ (liczba zasobów), oczekiwać by można, że program będzie wykonywać się bardzo długo. Naiwna implementacja synchronizacji procesów mogłaby być nawet rzędu $O(N*M) = O(10^{10})$, aczkolwiek jak zostało pokazane na przykładzie, czas uruchomienia programu był zaskakująco niski – **19.232s**!!
Podział zasobów nie tylko jest równomierny dla małych danych, ale również dla bardzo dużych. Co można zobaczyć w kolejnym przykładzie. Interesującą obserwacją jest to, że pod koniec standardowego wyjścia pojawia się dosłownie parę procesów o niskim zużyciu zasobów (np. No 895 = 17 i No 890 = 90), czego nie zaobserwowałem przeglądając wyniki w głąb.

```
janek@janek-z570:~/Studia/SO/Filozofowie$ time ./phi_race 1000 10000000 1 0 1
[N: 1000, M: 10000000, turn_on_plates: 1, turn_on_thinking: 0]
[print_off: true]

real    0m6.929s
user    0m4.820s
sys     0m19.232s
```

```
Philosoph No 520 has eaten 1048 dishes with summary thinking of 0.975 ms
Philosoph No 481 has eaten 1058 dishes with summary thinking of 0.419 ms
Philosoph No 891 has eaten 968 dishes with summary thinking of 0.689 ms
Philosoph No 173 has eaten 789 dishes with summary thinking of 0.538 ms
Philosoph No 489 has eaten 1082 dishes with summary thinking of 0.058 ms
Philosoph No 247 has eaten 990 dishes with summary thinking of 0.499 ms
Philosoph No 207 has eaten 675 dishes with summary thinking of 0.167 ms
Philosoph No 244 has eaten 995 dishes with summary thinking of 0.382 ms
Philosoph No 890 has eaten 99 dishes with summary thinking of 0.843 ms
Philosoph No 671 has eaten 1027 dishes with summary thinking of 0.065 ms
Philosoph No 423 has eaten 1017 dishes with summary thinking of 0.496 ms
Philosoph No 268 has eaten 1052 dishes with summary thinking of 0.523 ms
Philosoph No 136 has eaten 995 dishes with summary thinking of 0.382 ms
Philosoph No 672 has eaten 1046 dishes with summary thinking of 0.491 ms
Philosoph No 895 has eaten 77 dishes with summary thinking of 0.145 ms
Philosoph No 624 has eaten 1075 dishes with summary thinking of 0.361 ms
Philosoph No 14 has eaten 1084 dishes with summary thinking of 0.804 ms
Philosoph No 793 has eaten 1013 dishes with summary thinking of 0.418 ms
Philosoph No 925 has eaten 1046 dishes with summary thinking of 0.491 ms
Philosoph No 60 has eaten 994 dishes with summary thinking of 0.421 ms
Philosoph No 562 has eaten 1039 dishes with summary thinking of 0.041 ms
Philosoph No 720 has eaten 1014 dishes with summary thinking of 0.906 ms
Philosoph No 185 has eaten 738 dishes with summary thinking of 0.767 ms
janek@janek-z570:~/Studia/SO/Filozofowie$ ./phi_race 1000 1000000 1 1
```

Uwagi techniczne

Do rozwiązania załączony został plik **Makefile** stworzony tak, aby po wywołaniu polecenia **make** otrzymać pliki binarne, a po wywołaniu polecenia **make clean** usunąć je. W przeciwnym wypadku program powinien zostać skompilowany używając komendy: **$ gcc -std=gnu99 -pthread phi_race.c -o phi_race -lrt**. Program wykonywać można z argumentami, którego wywołanie wygląda następująco **$ ./phi_race ?N ?M ?turn_plates_on ?turn_thinking_on ?print_off**, gdzie *?N* i *?M* jest typu *int*, a reszta argumentów przyjmuje wartości 0 lub 1