# Teoria Współbieżności lab3

Jakub Koźlak

November 2021

## 1 Cel ćwiczenia

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

## 2 Zawartość archiwum

Rozwiązania znajdują się w dwóch katalogach. Oprócz nich znajdują się wykresy oraz dane otrzymane w czasie pomiarów różnych implementacji ( dla ilości filozofów $N = 5, 10, 100$).

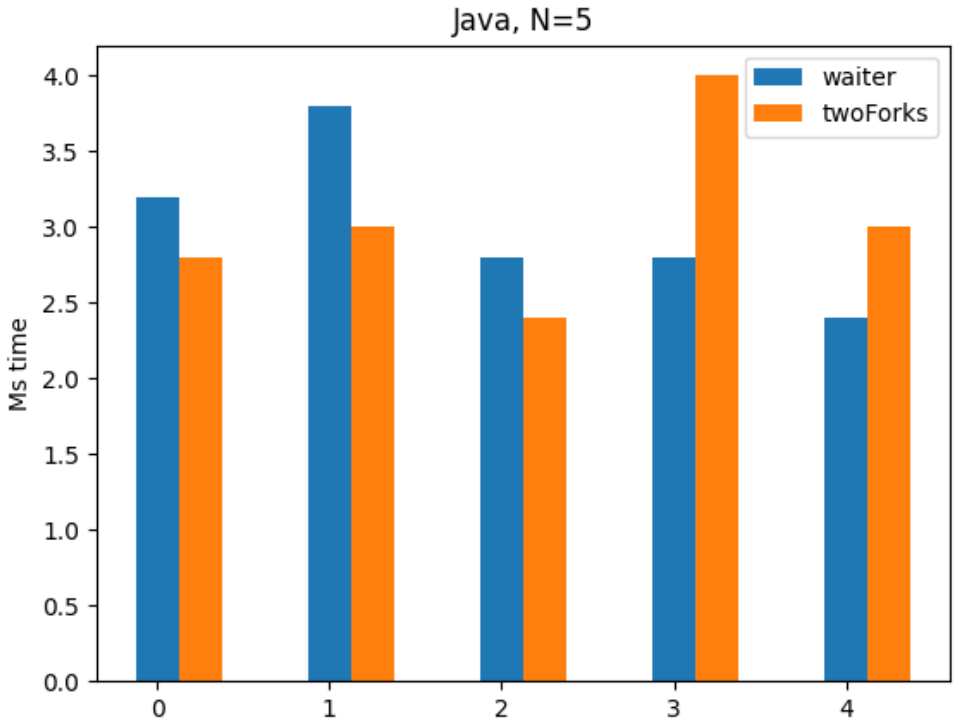
## 3 Java

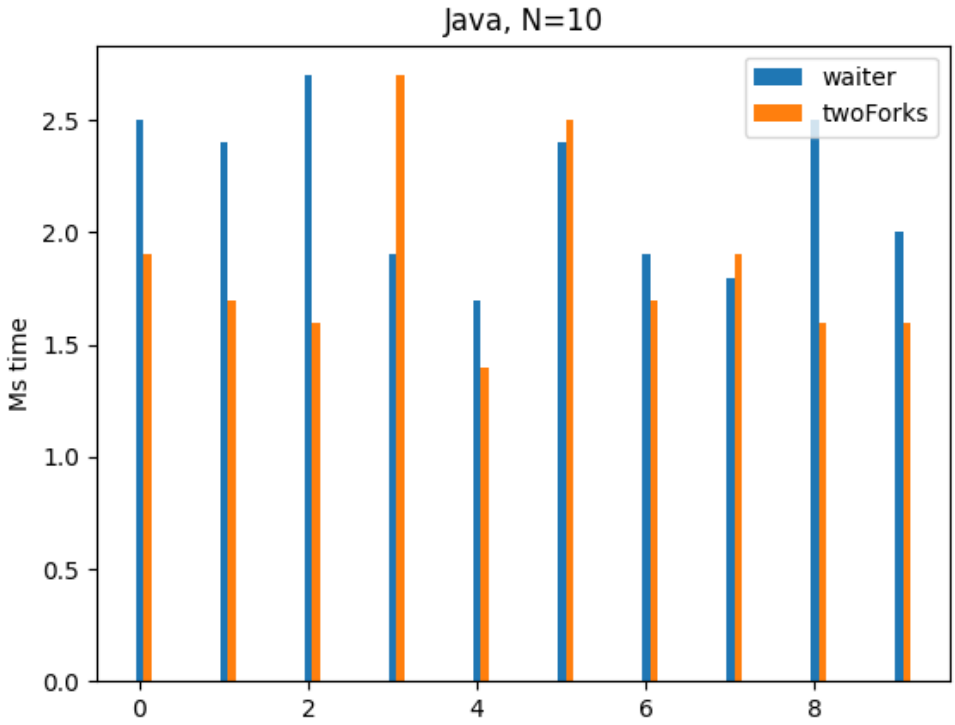
Implementacja dwóch podejść: z kelnerem oraz z jednoczesnym podnoszeniem widelców.

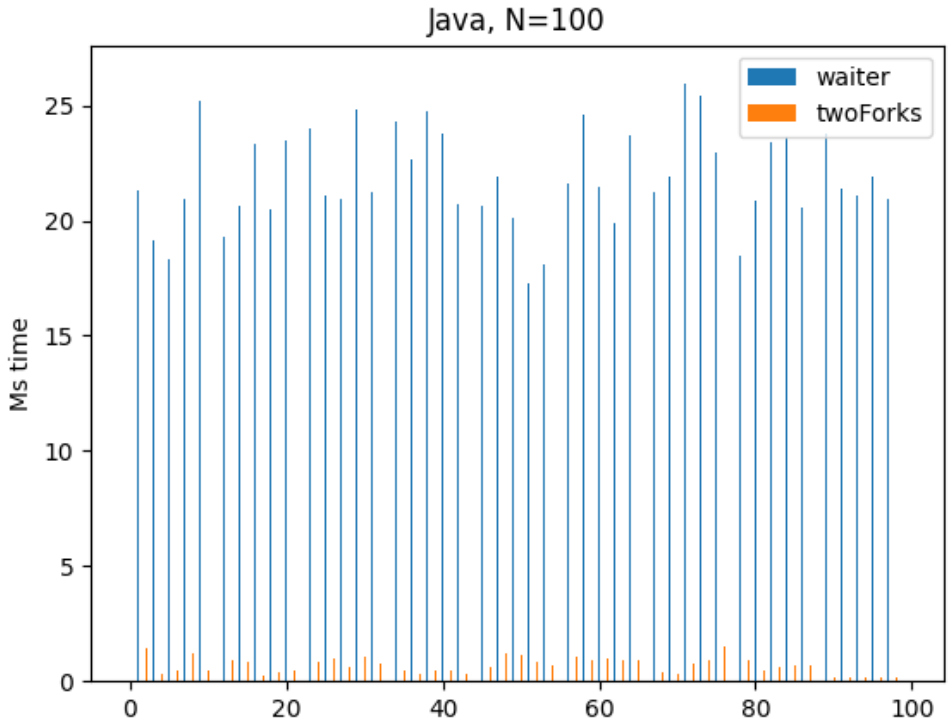
Z arbitrem - zewnętrzny arbiter (dalej: kelner), który ma swój semafor, a każdy filozof pyta kelnera o możliwość jedzenia, jeśli nie może spytać kelnera albo nie może jeść, będzie dalej próbował to zrobić.

Podnoszenie obu naraz - filozof albo podniesie dwa widelce, albo nie podnosi nic. Nie ma możliwości podniesienia dwóch semaforów w jednym kroku, więc w szczególności kiedy uda się podnieśc tylko jeden widelec, musimy go odłożyć.

Do implementacji obu podejść wykorzystałem tablicę semaforów, którą monitorowałem użycie widelców.


Java, N=5
waiter
twoForks
Ms time
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4

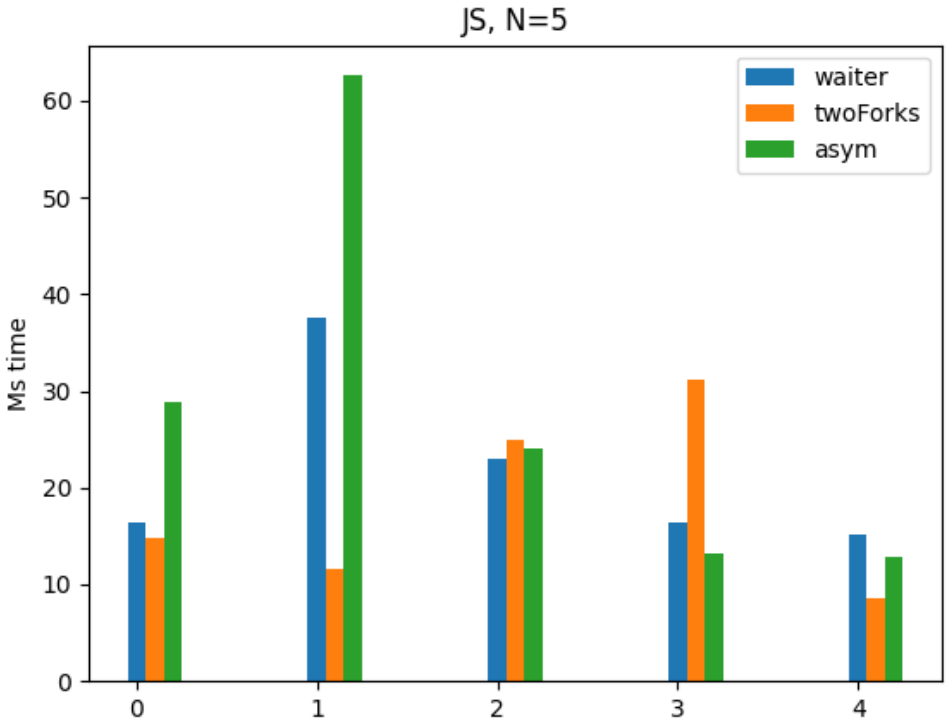

Java, N=10
waiter
twoForks
Ms time
0.0
0.5
1.0
1.5
2.0
2.5
0
2
4
6
8

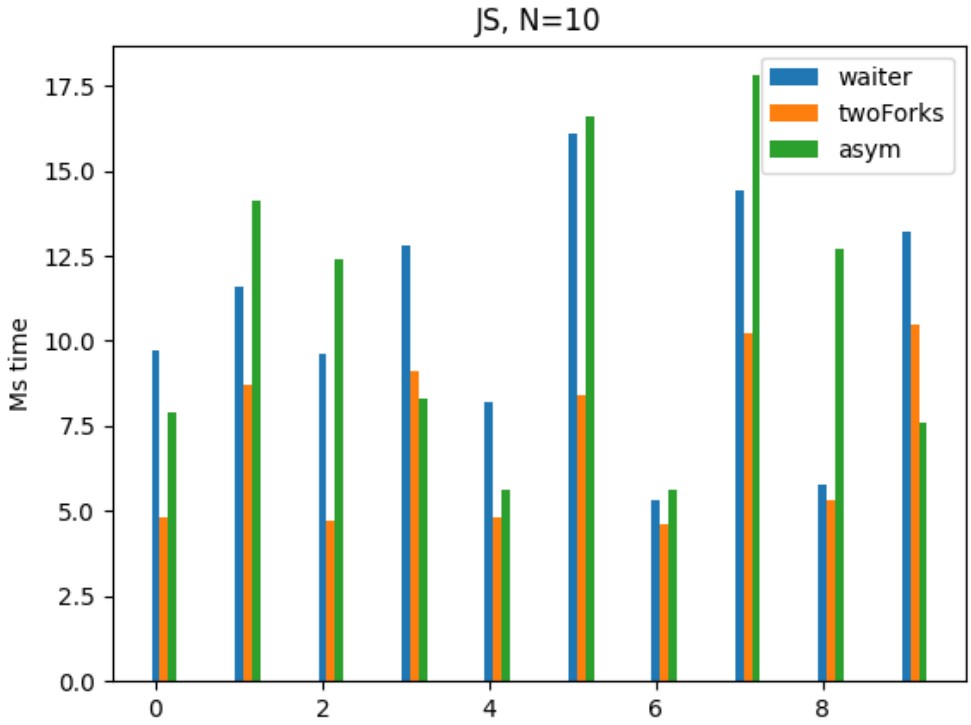


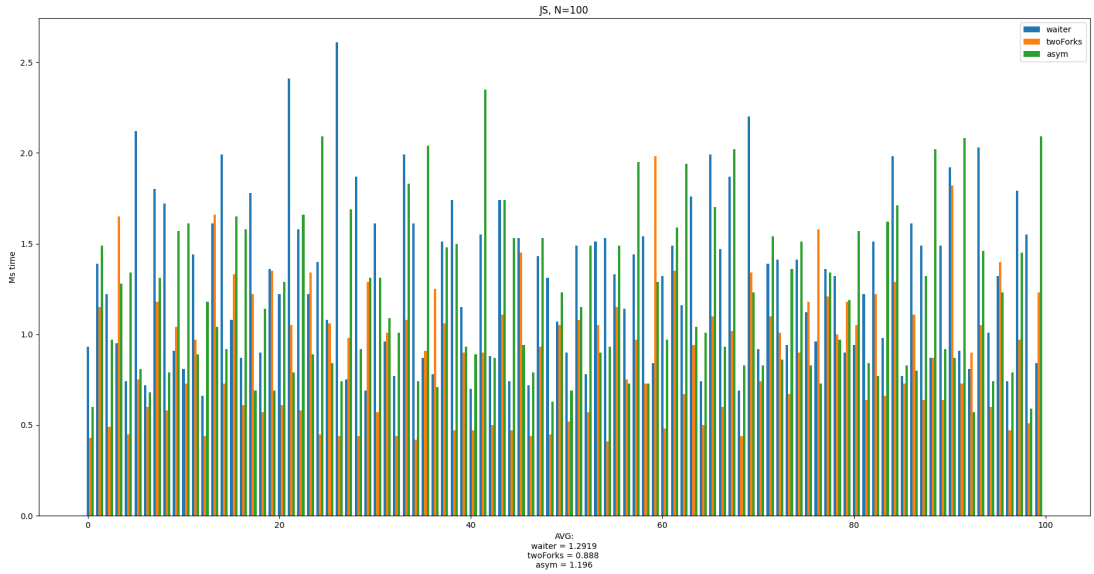

Widać, że rozwiązanie z kelnerem daje dużo gorsze rezultaty niż podnoszenie dwóch widelców, szczególnie dla dużego N. Najbardziej intuicyjne z Javowych rozwiązań okazuje się być tym wolniejszym.

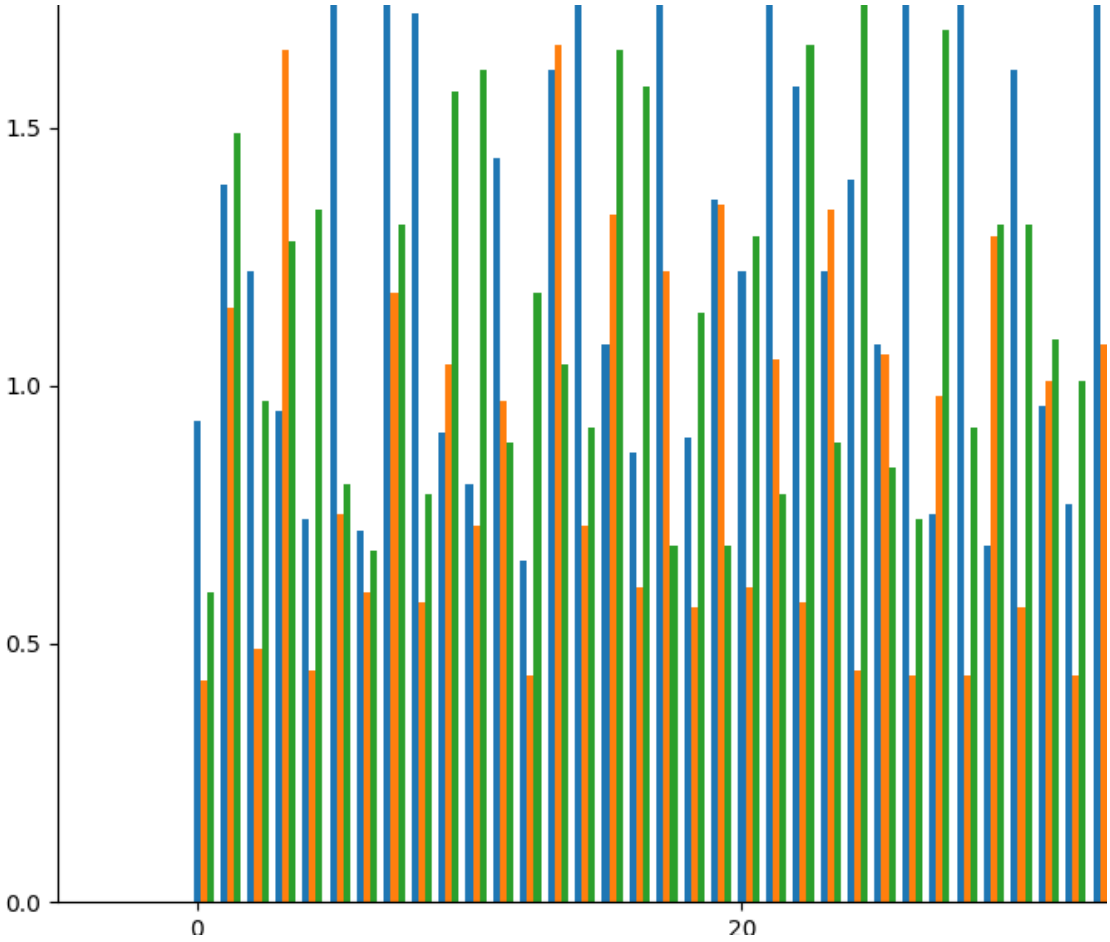
# 4 JavaScript

Naiwne - takie, w którym filozofowie podnoszą widelec jeśli mogą i czekają na następny. Dochodzi tutaj do zakleszczenia - nie zostanie ani jeden wolny widelec.
Asymetryczne - filozof z parzystym numerem podnosi najpierw prawy widelec, z nieparzystym najpierw lewy.
Z arbitrem, podnoszenie obu naraz - jak w punkcie powyżej.
Algorytm $BEB$ zrealizowany za pomocą $setTimeout$, gdzie czas oczekiwania zwiększa się z każdą próbą (zwiększa się zasięg losowania czasu oczekiwania).


JS, N=5
waiter
twoForks
asym
Ms time
0
10
20
30
40
50
60
0
1
2
3
4


JS, N=10
waiter
twoForks
asym
Ms time
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
2
4
6
8


JS, N=100
waiter
twoForks
asym
Ms time
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
AVG:
waiter = 1.2919
twoForks = 0.888
asym = 1.196




Średnie czasy dla $N = 100$ wyniosły:
$waiter = 1.2919$,
$twoForks = 0.88$,
$aSym = 1.196$

# 5 Wnioski

- Rozwiązanie z arbitrem wydaje się najgorsze - chociaż wcale nie jest to takie jednoznaczne. Przy wzroście ilości filozofów nadal posiadamy jednego (!) kelnera, który stanowi swego rodzaju wąskie gardło dla szybkości działania.
- Strategia podnoszenia dwóch widelców naraz, chociaż ma możliwość zagłodzenia procesu, nie robi tego i czasy dostępów są krótsze niż w przypadku innych metod.
- Java pracując na wielu wątkach może osiągać (przynajmniej dla mnie)

imponujące czasy działania, nawet pomimo potrzeby synchronizacji. JavaScript posiada natomiast jeden wątek, w którym kolejkuje kolejne instrukcje.

- Asymetryczne, JavaScript- niewielkie różnice w czasie działania w porównaniu do arbitra wynikają prawdopodobnie z błędów implementacji (która to sprawiła wiele kłopotów, a kod zmienił się w plątaninę funkcji i odwołań).
- Do różnicy w wynikach między środowiskami należy dorzucić jeszcze losowe działanie algorytmu $BEB$.