Sprawozdanie TW – Problem pięciu filozofów

Gabriel Cyganek

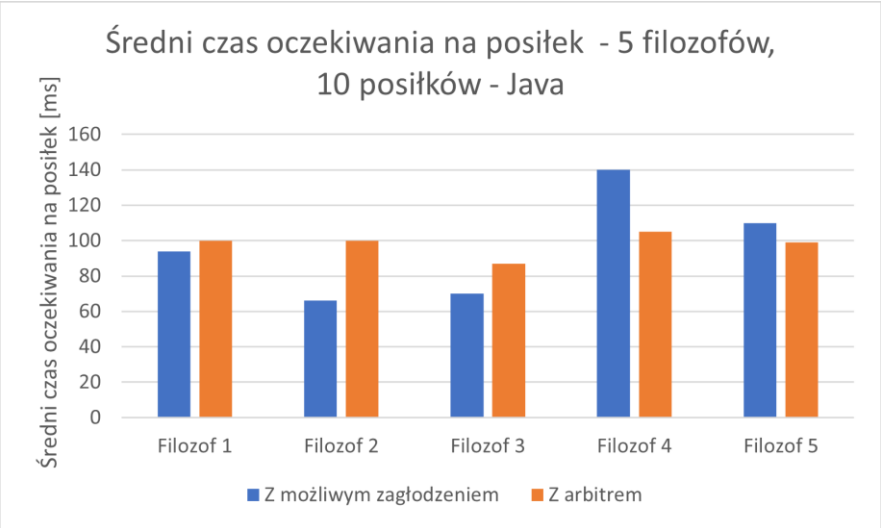
1. Dane techniczne

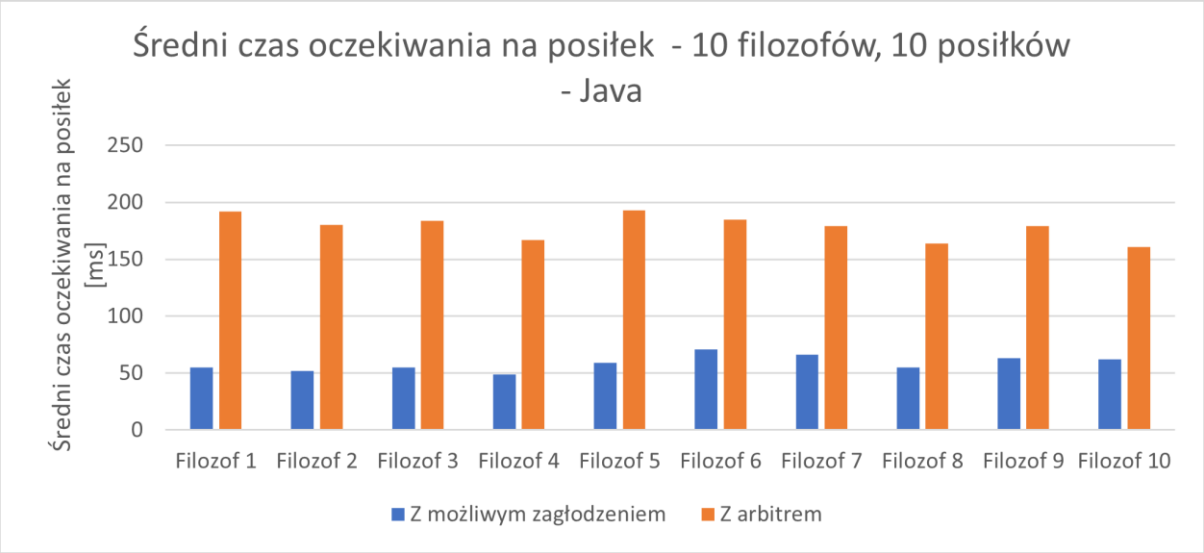
Eksperymenty zostały przeprowadzone na systemie Windows 10 64-bit z node.js v16.13.0 oraz OpenJDK 16, na procesorze Intel Core i5-7300HQ 2.50 GHz.

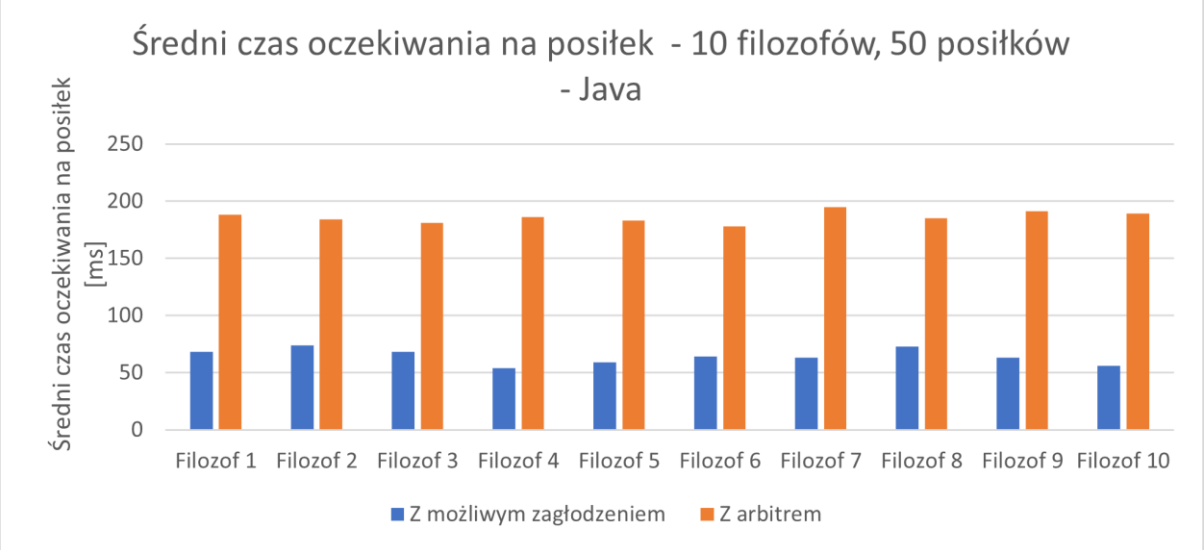
2. Programowanie wielowątkowe w Javie

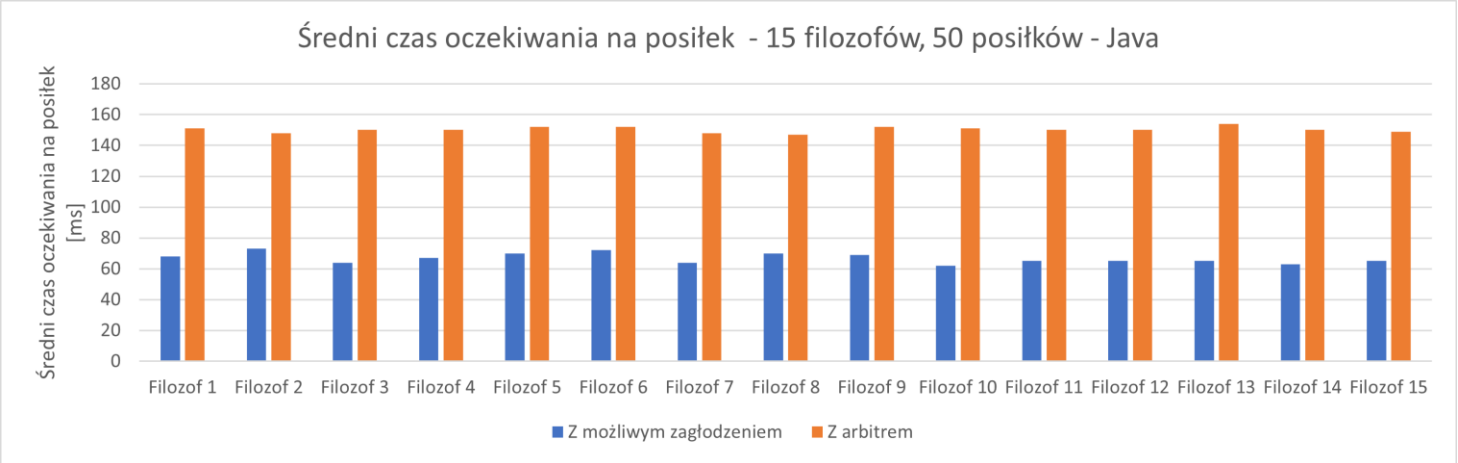
Założyłem, że filozofowie przed każdym posiłkiem myślą przez losowy czas z zakresu 20-80ms, a następnie próbują wziąć widelce i rozpocząć jedzenie, które trwa 100ms. Przy wariancie z możliwością zagłodzenia (klasa StarvingPhilosopher) filozof próbuje wziąć lewy widelec, a następnie, jeśli jest to możliwe, bierze prawy. W przeciwnym przypadku odkłada lewy widelec i próbuje ponownie aż do otrzymania obu widelców. W wersji z arbitrem (klasa LimitedSeatsPhilosopher) filozof najpierw zajmuje miejsce przy stole (klasa Table), jeśli dla N filozofów nie zostało jeszcze zajętych N-1 miejsc. W przeciwnym przypadku czeka na miejsce, a następnie próbuje wziąć najpierw lewy, a następnie prawy widelec i rozpoczyna jedzenie. Aby przeprowadzić eksperyment należy uruchomić metodę main() z klasy Demo i wprowadzić odpowiednie argumenty wejściowe.

Przykładowe eksperymenty:









Średnie czasy oczekiwania są wyższe dla wariantu z arbitrem. Jest to koszt, jaki ponosimy, aby nie otrzymać zagłodzenia któregoś z filozofów. Analizując wyniki eksperymentu zauważyłem, że przy wariancie z możliwym zagłodzeniem niektórzy

filozofowie jedli cały czas do zakończenia wszystkich swoich posiłków, aby dopiero potem dopuścić do jedzenia pozostałych, którzy mogli już bez czekania jeść kolejne posiłki. Widać zatem, że nie jest to najbardziej sprawiedliwy wariant i w skrajnym przypadku filozof mógłby czekać na zjedzenie pierwszego posiłku do czasu, aż obaj jego sąsiedzi zjedzą wszystkie swoje posiłki.

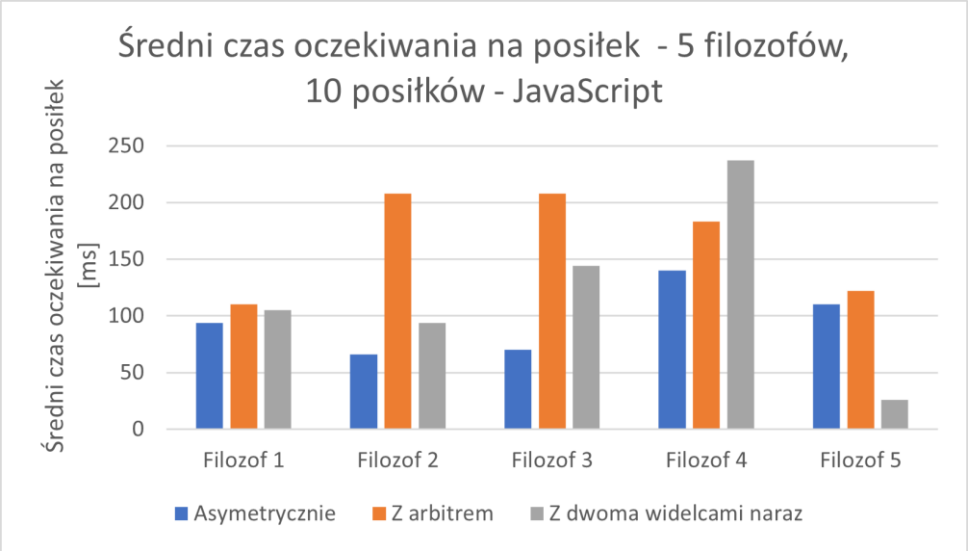
3. Programowanie asynchroniczne w środowisku uruchomieniowym *node.js*

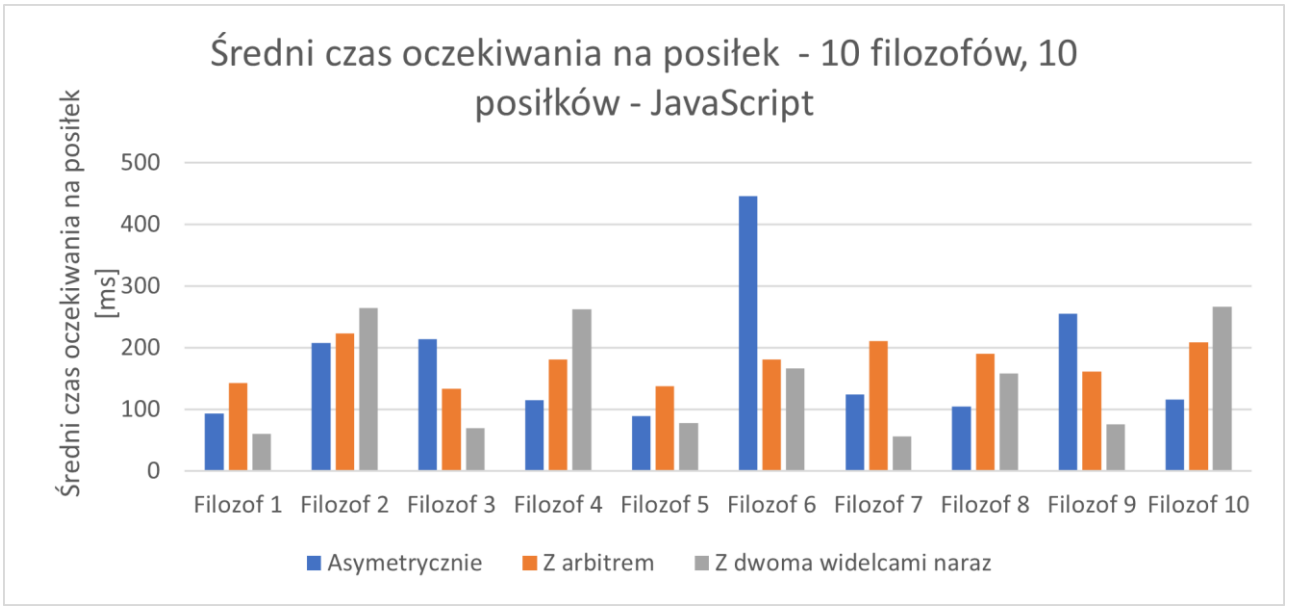
Podobnie jak dla rozwiązań napisanych w języku Java, tak i tutaj zastosowałem zasadę myślenia przez losowy czas z zakresu 20-80ms, a następnie próby wzięcia widelców i jedzenia przez 100ms. Stosowałem głównie funkcje zwrotne oraz wykonywanie funkcji po określonym czasie przez funkcję *SetTimeout()*.

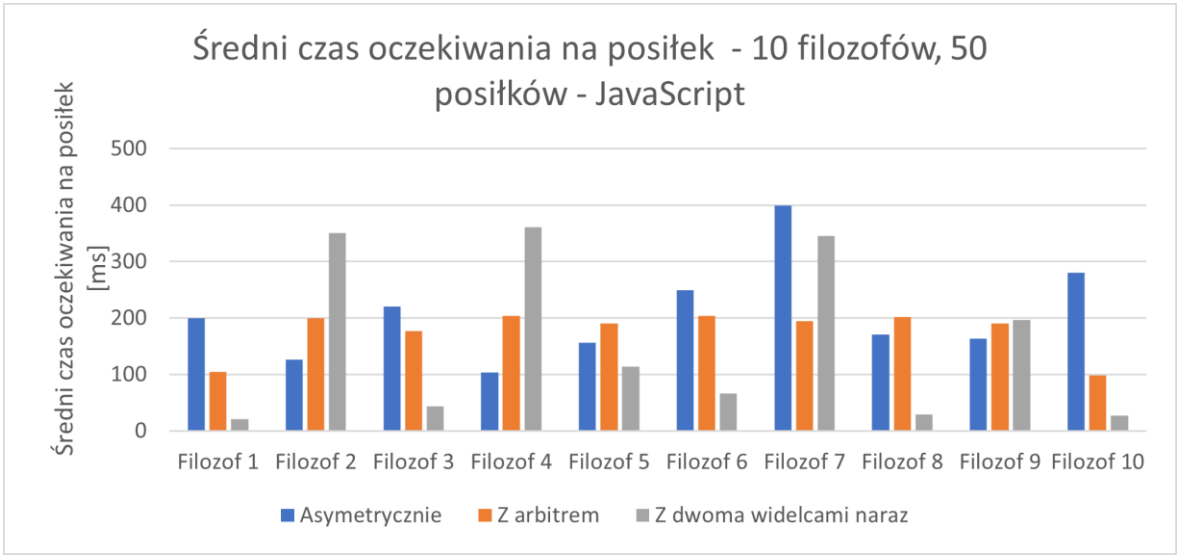
W wariancie z kelnerem, aby nie dopuścić do sytuacji, w której niepożądane osoby dostaną się do stołu sprawdzam nie tylko zajęte widelce, ale też widelce, które mogą być zajęte przez osoby już dopuszczone do stołu. W związku z ubogim i nie do końca według mnie zrozumiałym opisem na polskiej Wikipedii postanowiłem, że w moim programie kelner będzie dopuszczał do stołu osoby, dla których są oba widelce wolne, a w przeciwnym przypadku zostaną one dołączone do kolejki, która jest ponownie sprawdzana przy każdym zwolnieniu miejsca przez filozofa, który właśnie zakończył dany posiłek.

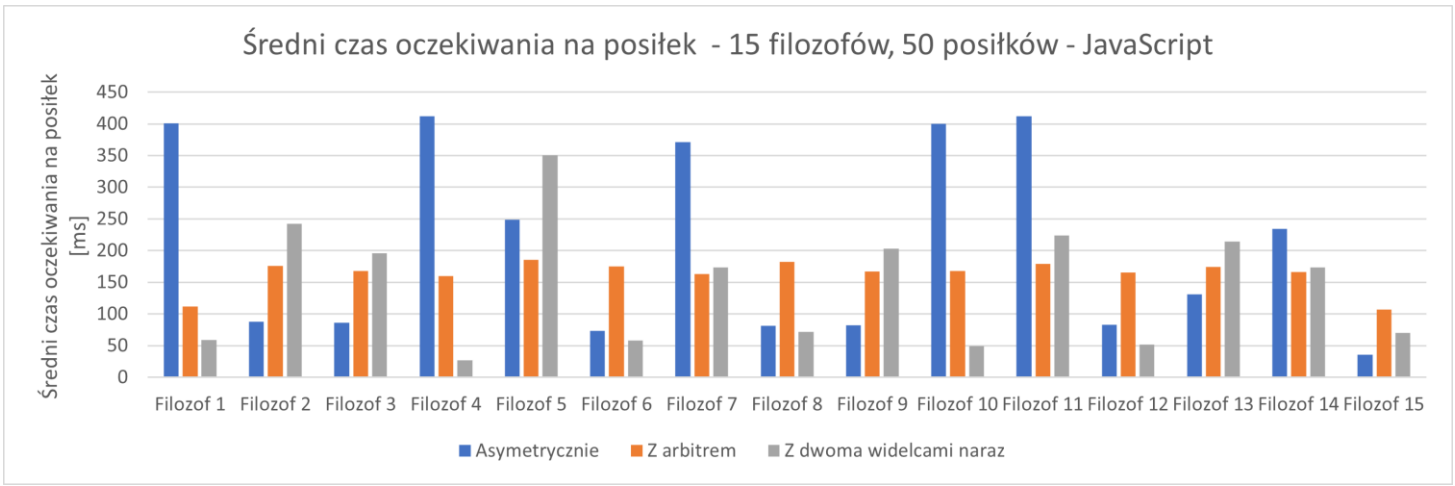
Aby przeprowadzić eksperyment należy uruchomić plik *philosophers.js* komendą *node philosopher.js*, a następnie wprowadzić odpowiednie argumenty wejściowe.

Przykładowe eksperymenty:









Ponownie wariant z arbitrem (mimo, iż jest on jeśli chodzi o algorytm rozwiązania nieco inny niż w Javie) wydaje się być najbardziej sprawiedliwy, widać to zwłaszcza na

wykresie z eksperymentem dla 15 filozofów i 50 posiłków. Wariant z dwoma widelcami naraz a także wariant asymetryczny posiadają filozofów, którzy czekają zdecydowanie więcej od pozostałych. Często zostają oni dopuszczani do jedzenia dopiero wtedy, kiedy sąsiadujący filozofowie zakończą wszystkie swoje posiłki. Ta różnica jest tym bardziej widoczna ze względu na wykorzystanie algorytmu BEB. Zdarza się, że gdy filozof mógłby już otrzymać oba widelce, nie otrzymuje ich, gdyż musi odczekać pewien czas przed ponownym zapytaniem o widelec. Gdy ten czas minie, widelce są znów zajęte, a filozof musi czekać dwa razy dłużej niż ostatnio przed kolejnym zapytaniem. W wariancie z arbitrem filozof zostaje dopuszczony do stołu, gdy oba jego widelce są wolne, zatem nie ma potrzeby ciągłego pytania o widelec po zajęciu miejsca przy stole.

4. Programowanie asynchroniczne, a programowanie wielowątkowe

Z uwagi na to, że programy asynchroniczne i wielowątkowe były wykonywane w różnych środowiskach ciężko jest je ze sobą porównywać, jednak na podstawie wykresów można zauważyć, że w wersji JavaScriptowej zdarzało się, że filozofowie czekali średnio nawet kilkaset ms, gdzie w wersji Javowej maksymalne wartości średnie oscylują w okolicach 200ms. Można przypuszczać, że ma to związek z tym, że w Javie, w której możemy wykonywać wiele wątków na różnych procesorach i ich rdzeniach jednocześnie korzystaliśmy z mechanizmów synchronizacji, a w JavaScriptcie, gdzie mamy Event Loop działający na pojedynczym wątku i callbacki obsługiwane w odpowiedniej kolejności, z BEB, które momentami powodowało znaczne wzrosty czasu oczekiwania filozofów, gdyż mogli oni przegapić moment, kiedy widelce były dostępne, ponieważ nie minął jeszcze czas przed kolejnym zapytaniem o nie. W skrajnym przypadku mogło dość do sytuacji, w której filozof wiele razy mógł wziąć widelec, jednak nie trafiał z ponownym zapytaniem w odpowiednim momencie i otrzymał go dopiero po tym, gdy sąsiedni filozof zakończył wszystkie swoje posiłki.