Teoria Współbieżności
Zadanie domowe
Mateusz Skowron

Zawartość archiwum
W głównym katalogu archiwum znajdują się dwa podkatalogi *Java* oraz *JavaScript* zawierające rozwiązania problemu w danych językach programowania dla każdego z przypadków (asymetryczne, naiwne, z arbitrem oraz podnoszenie dwóch widelców jednocześnie). W obu podkatalogach znajduje się dodatkowo plik bashowy *run-and-draw*, który w przypadku wersji JavaScriptowej odpala program dla różnych parametrów (5, 10, 15 filozofów, gdzie każdy filozof zjada posiłek 10, 15, 25 razy) i na podstawie wyników generuje wykres używając *gnuplota*, natomiast w wersji Javowej pojawiło się u mnie dużo problemów związanych z kompilacją rozwiązania używając skryptu bashowego, a zatem zdecydowałem się skompilować każdy przypadek ręcznie i zapisać wyniki do pliku, a następnie używając *gnuplota* sporządziłem wykresy na podstawie otrzymanych wyników.Stąd plik *run-and-draw* dla wersji Javowej zawiera jedynie kod rysowania wykresu na podstawie danych z plików. Wykresy również znajdują się w każdym z podkatalogów.

Implementacja wersji JavaScript
Program uruchamiamy poleceniem

$node main.js

Zostajemy zapytani o liczbę filozofów oraz ile razy każdy z nich będzie spożywał posiłek. Na wyjściu otrzymujemy średnie wyniki czekania na dostęp do widelców dla każdego z filozofów dla 3 różnych strategii symulacji odpowiednio: asymetrycznej (Asym), z podnoszeniem obu widelców na raz (BothForksAtOnce) oraz z kelnerem (Conductor). Kluczowa w implementacji okazała się funkcja $setTimeout()$, która po określonym przez drugi parametr czasie przekazany callback dodaje do stosu czekających ramek na wykonanie przez event loop. Funkcja $acquire()$ zarówno dla poszczególnego widelca jak i dla kelnera korzysta z algorytmu *BEB,* czyli przed pierwszą próbą podniesienia widelca Filozof odczekuje 1ms, a jeśli to się nie uda, czas zwiększany jest dwukrotnie i po raz kolejny próbuje podnieść widelec, itd.

Implementacja wersji Java
Program uruchamiamy poleceniem

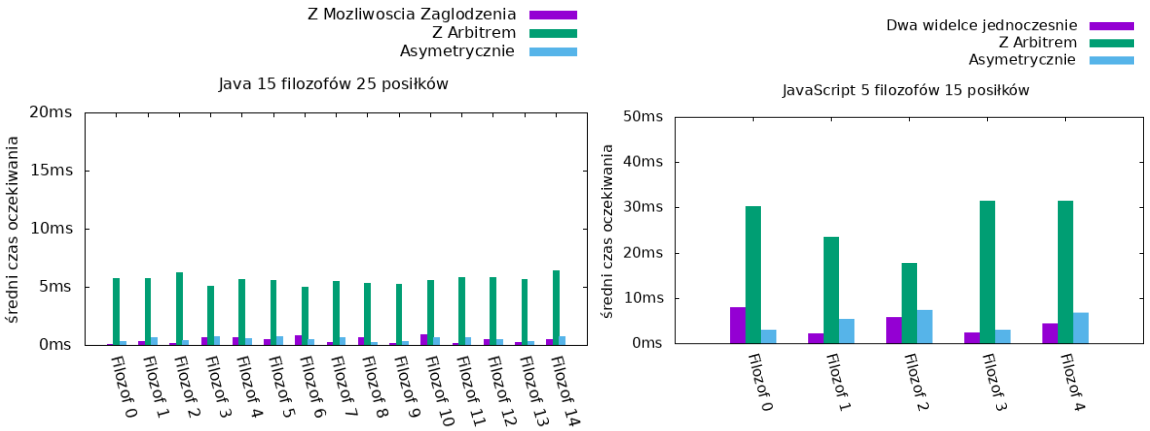
$gradlew run —args=”liczba_filozofow liczba_posilkow”

Program zostanie uruchomiony dla każdej z 3 strategii symulacji odpowiednio: asymetrycznej, z arbitrem (kelnerem) oraz z możliwością zagłodzenia. Implementacja każdej ze strategii znajduje się w odpowiednim podkatalogu odpowiednio: *asymetryczne, zarbitrem* oraz *zmozliwosciazaglodzenia*. W każdym podkatalogu znajdują się 3 klasy:

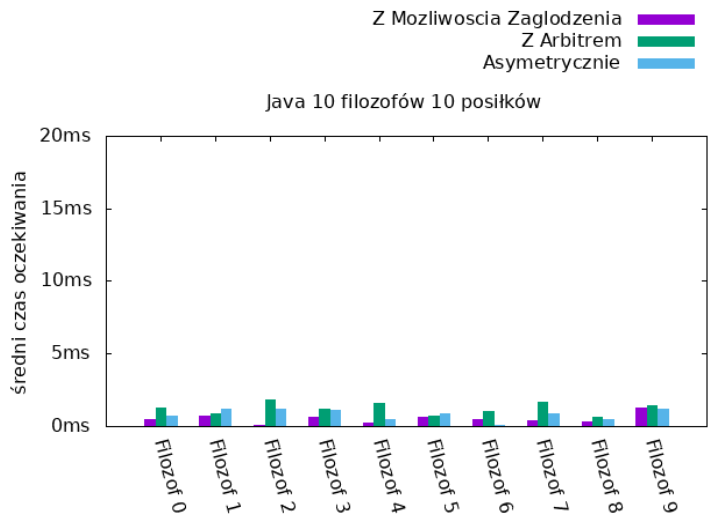
- *Filozof* - Klasa reprezentująca filozofa, gdzie lewy oraz prawy widelec to semafory binarne. Rozszerza ona klasę *Thread*, aby każdego z filozofów można było odpalić w osobnym wątku. Nadpisana z klasy *Thread* metoda *run* implementuje główną pracę filozofa.
- *Stol* - Klasa zawierająca tablicę widelców oraz w przypadku strategii z arbitrem posiada dodatkowe pole *arbiter*, które jest semaforem licznikowym.
- *Symulacja* - Klasa, która uruchamia symulację dla danej strategii, a następnie wypisuje średnie czasy oczekiwania filozofów.

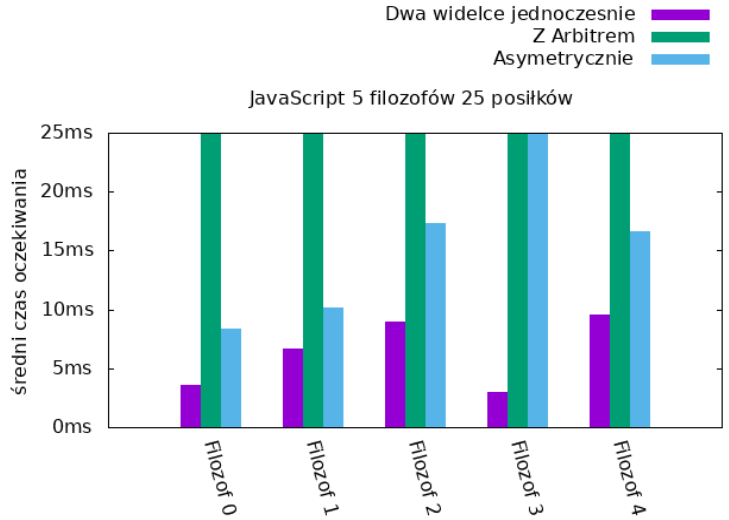
W klasie *Main* uruchamiana jest symulacja dla każdej ze strategii w wątku, na który następnie oczekujemy aż się zakończy i uruchamiamy kolejną symulację dla następnej strategii. Liczba filozofów oraz jedzonych przez nich posiłków pobierana jest z argumentów, z którymi uruchomiono program.

Wyniki i wnioski
Patrząc na średnie czasy oczekiwania łatwo zauważyć, że rozwiązanie z arbitrem, które teoretycznie jest bardzo dobre, nie daje optymalnych wyników, gdyż średnie czasy oczekiwania są wyższe od pozostałych strategii, co jest szczególnie widoczne jeśli zwiększymy liczbę filozofów. Jest ono natomiast bardziej sprawiedliwe (co jest doskonale widoczne w przypadku rozwiązania Javovego), ponieważ żaden filozof nie ma dużo gorszej/lepszej sytuacji w porównaniu do innych jak w innych strategiach.



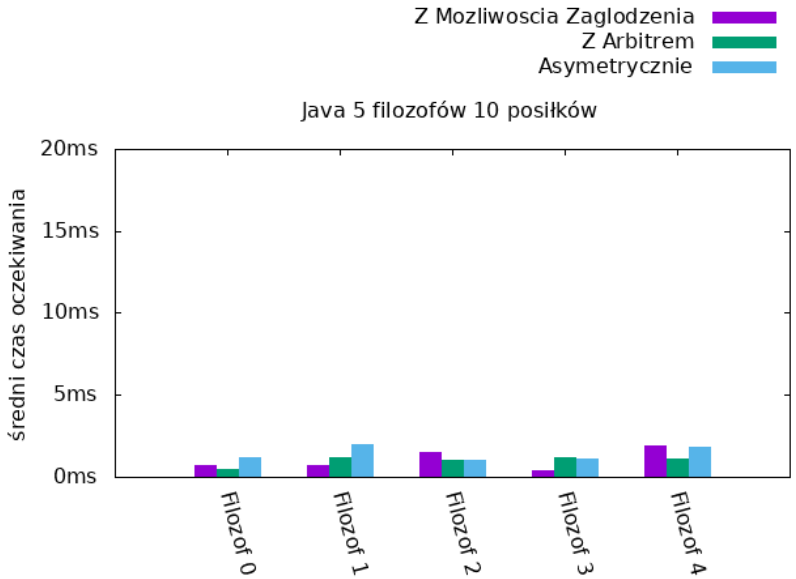
Obserwując wykresy można dostrzec przewagę rozwiązania z podnoszeniem dwóch widelców jednocześnie nad rozwiązaniem z arbitrem oraz asymetrycznym. W teorii jest to rozwiązanie najbardziej podatne na zagłodzenie, ale szansa na to jest bardzo niewielka.

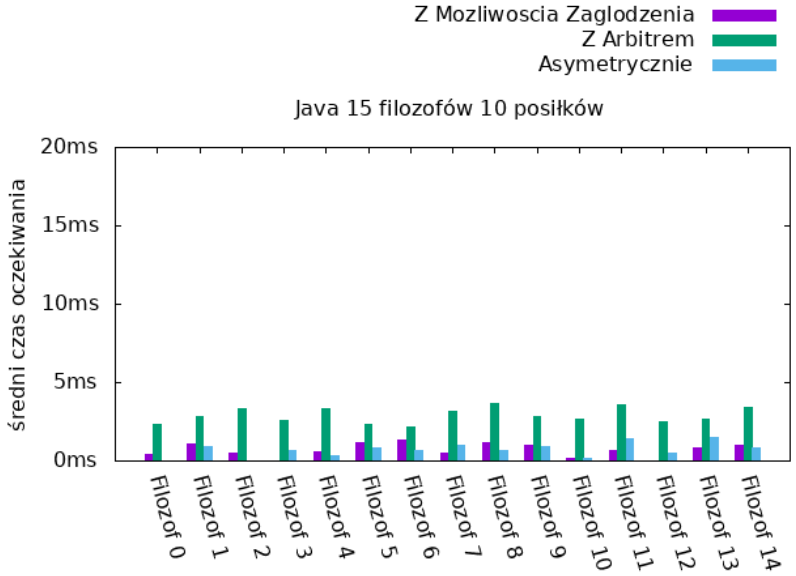




Na wykresach widać, że opisane wyżej zjawisko zachodzi zarówno w przypadku implementacji Javovej jak i JavaScriptowej.

Poprzez zwiększanie liczby filozofów widoczny jest znaczny wzrost w czasie oczekiwania w rozwiązaniu z kelnerem.
Średnie czasy oczekiwania w dwóch pozostałych strategiach nie wydają się zależeć od liczby filozofów.





Zwiększając liczbę filozofów średnie czasy oczekiwania pomiędzy implementacjami nie różnią się aż tak bardzo. Uwagę natomiast przykuwa fakt, iż strategia z arbitrem pomiędzy wersją Javovą, a JavaScriptową bardzo różni się pod względem czasowym. Wersja JavaScriptowa ma znacznie większe średnie czasy oczekiwania. Ze względu na różne środowiska ciężko podać dokładną przyczynę tego zjawiska.