# Teoria współbieżności

## Problem producent-konsument – porównanie implementacji z wykorzystaniem aktywnego obiektu z implementacją na trzech lockach

Mateusz Łopaciński

## 1. Opis implementacji AO (Active Object):

W implementacji wzorca Active Object wykorzystuję klasę Proxy, pozwalającą na pobieranie oraz dokładanie elementów do bufora. Jest to klasa, która pośredniczy w komunikacji wątków ze schedulerem i odpowiada za dokładanie do kolejki oczekujących na wykonanie zadań w schedulerze oraz pobieranie zadań z tej kolejki.

Scheduler uruchomiony jest w osobnym wątku. Jego zadaniem jest obsługa kolejki zadań oczekujących na wykonanie. Aby nie dopuścić do zakleszczenia, zadania, których nie można obecnie wykonać, ze względu na zbyt dużo/mało elementów w buforze, umieszczone zostają w osobnej kolejce. Ta kolejka traktowana jest z wyższym priorytetem, przez co najpierw, jeżeli jest to możliwe, wykonywane są zadania z tej kolejki, a dopiero później nowe zadania z pierwszej kolejki.

Jeżeli pierwszym czekającym zadaniem w drugiej kolejce jest zadanie konsumpcji, którego nie możemy wykonać ze względu na niewystarczającą liczbę elementów w buforze i jednocześnie nowe zadanie z pierwszej kolejki jest zadaniem konsumpcji, aby nie dopuścić do zagłodzenia, umieszczamy nowe zadanie również w drugiej kolejce, nawet wtedy, gdy możliwe jest jego natychmiastowe wykonanie. W ten sposób zachowujemy kolejność obsługiwania konsumentów zgodną z kolejnością zgłaszania przez nich chęci pobrania elementów z bufora, przez co nie dopuścimy do sytuacji, w której pewien wątek konsumenta będzie otrzymywał zasoby rzadziej od pozostałych. Analogicznie sytuacja wygląda w przypadku producentów.

Podczas oczekiwania na zwrócenie wyniku przez MessageFuture, wątki są obciążane wykonywaniem w pętli obliczeń (dodawania sumy sinusa i cosinusa do pewnej zmiennej).

## 2. Opis przeprowadzonych eksperymentów:

Zarówno implementacja problemu producenta-konsumenta z wykorzystaniem wzorca Active Object oraz implementacja w oparciu o 3 locki zostały uruchomione dla stałego czasu rzeczywistego, wynoszącego 10 sekund. Oba programy zostały uruchomione kolejno dla 2, 4, 6, 8, 10, 15, 20, 25, 30, 40 oraz 50 wątków[1]. Testy zostały przeprowadzone przy buforze o wielkości ustalonej na 1000 elementów. Wątki konsumentów konsumują porcje losowej wielkości od 1 do 500 elementów, natomiast wątki producentów, produkują porcje losowej wielkości, liczące od 1 do 500 elementów.

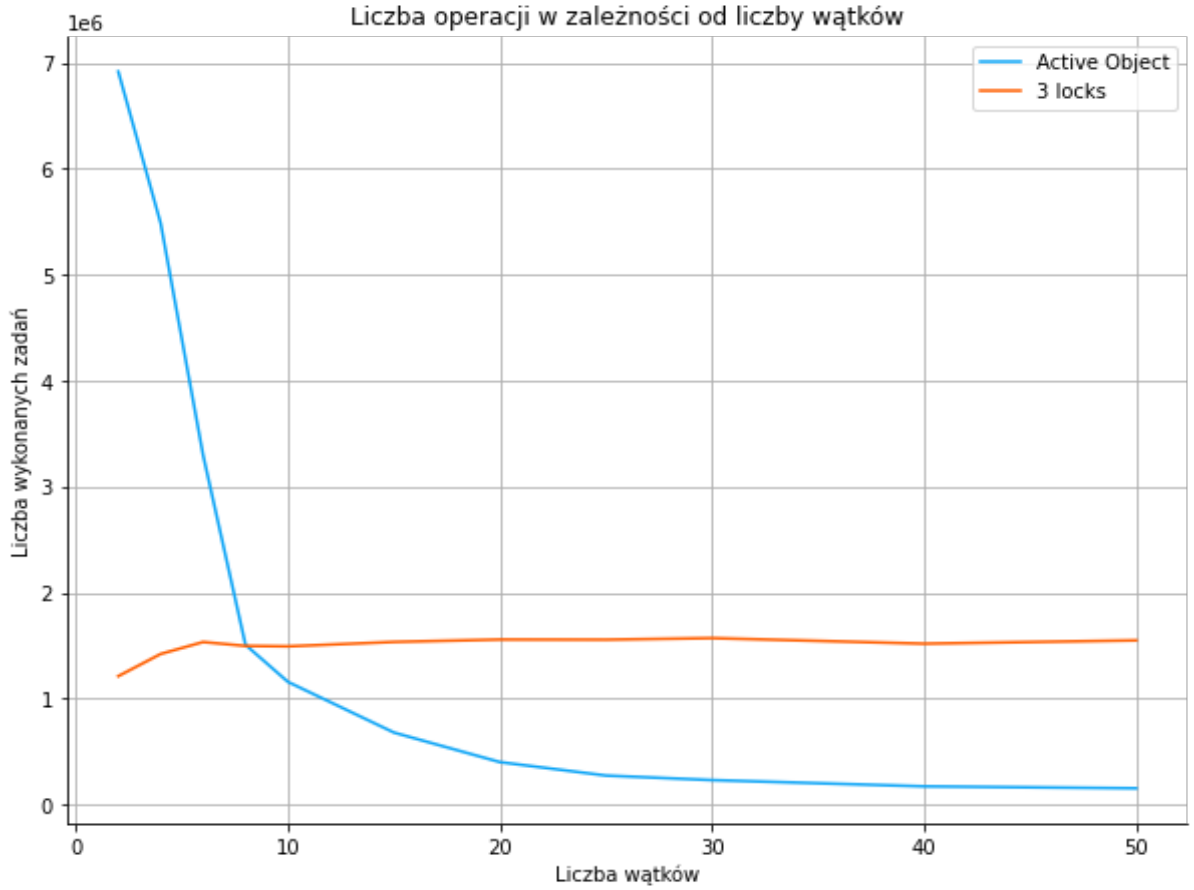
[1] Jest to sumaryczna liczba producentów oraz konsumentów podzielona po równo na wątki obu kategorii. W przypadku, gdy liczba wątków jest nieparzysta, liczba konsumentów jest o 1 większa od liczby producentów. We wzorcu wątek shedulera nie jest liczony do tej puli wątków.

W celu zapewnienia bardziej dokładnych wyników, dla każdego zestawu parametrów, testy zostały powtórzone 10-krotnie, a następnie wyniki testów zostały uśrednione.

Po upływie 10 sekund czasu rzeczywistego, wszystkie wątki (producenci oraz konsumenci) zostały zatrzymane, po czym zmierzony został sumaryczny czas CPU, a także czas CPU przypadający na 1 wątek, liczba wykonanych operacji na buforze oraz liczba operacji przypadająca na jednostkę czasu CPU. Wyniki zostały zebrane w następnym punkcie sprawozdania.

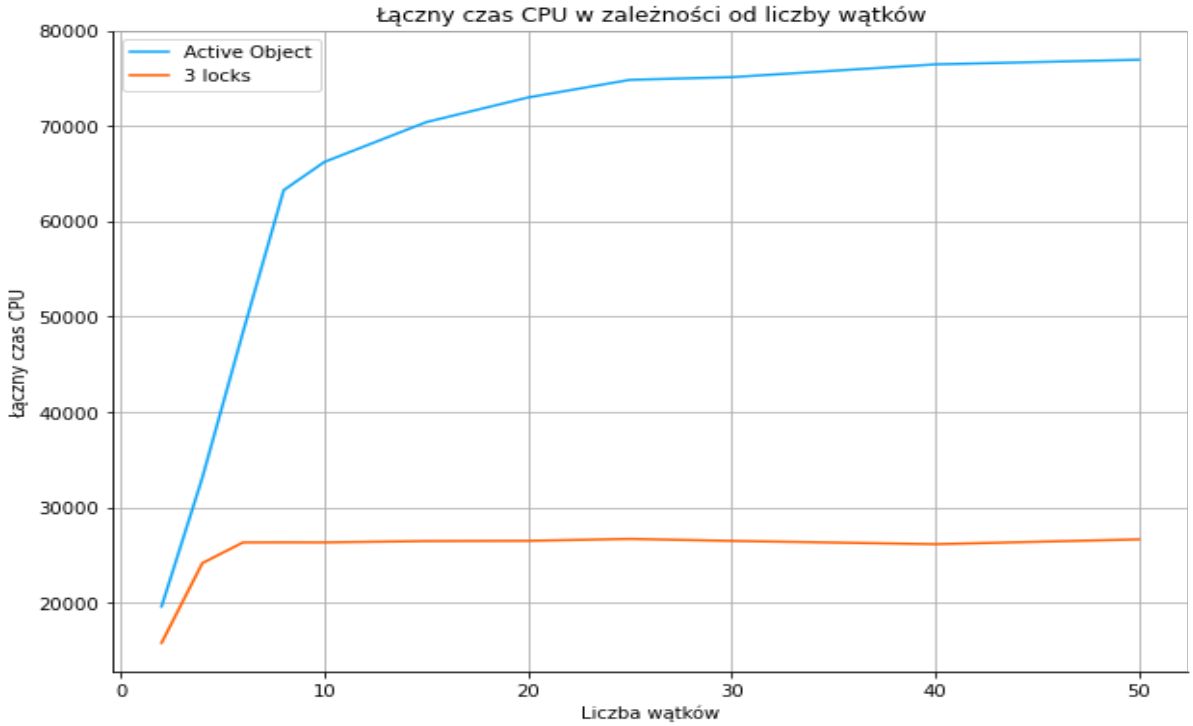
# 3. Wyniki testów

## 3.1. Liczba operacji w zależności od liczby wątków



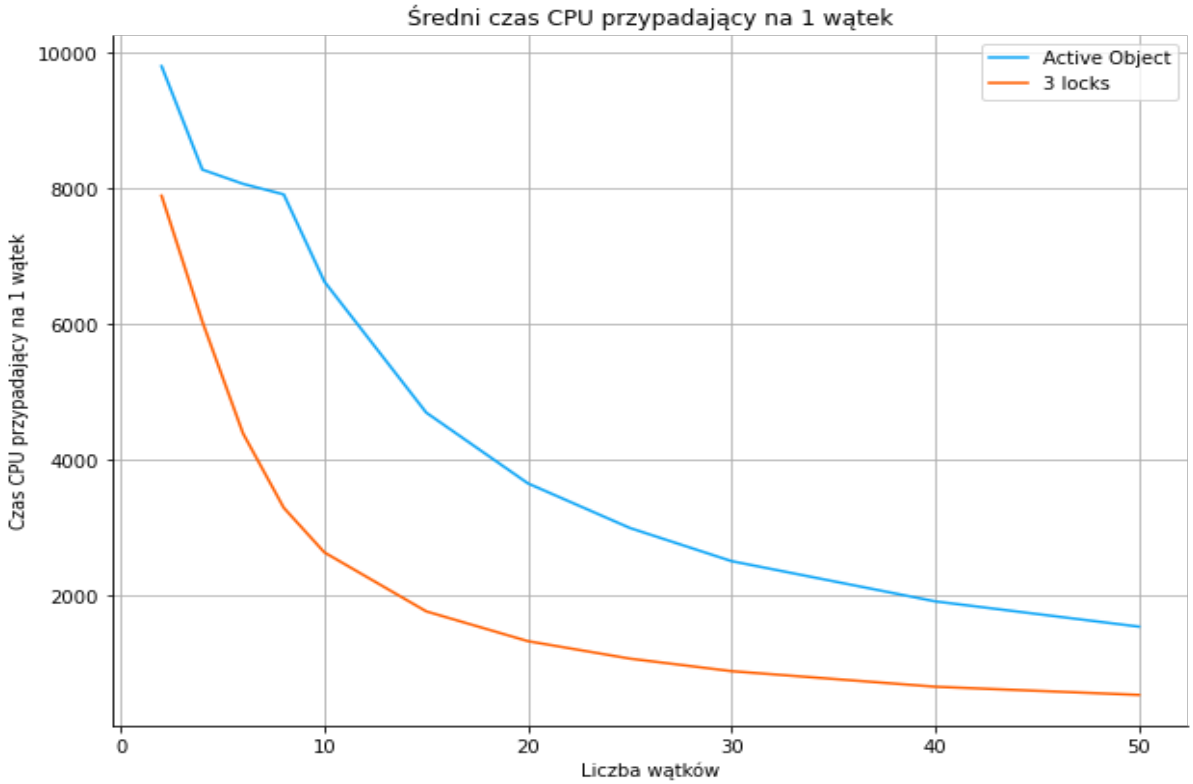

*3.1. Wykres liczby operacji w zależności od liczby wątków*

## 3.2. Łączny czas CPU w zależności od liczby wątków



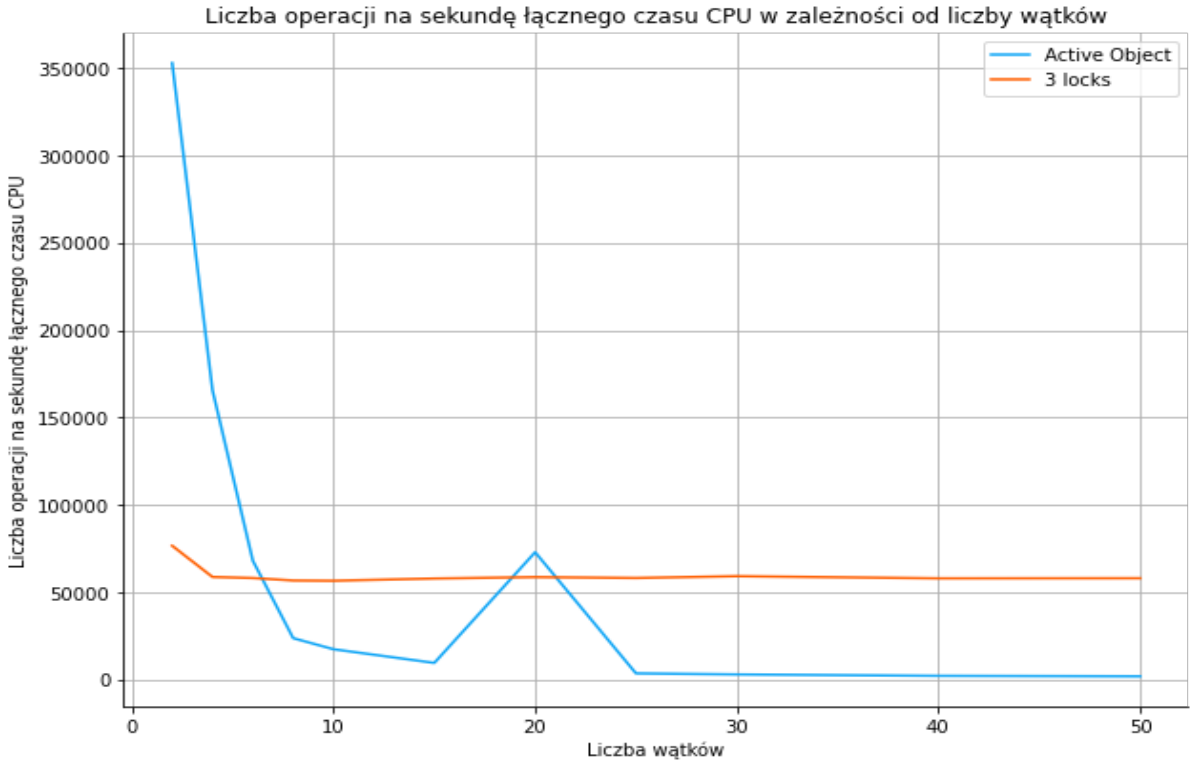

*3.2. Wykres zależności łącznego czasu CPU od liczby wątków [ms]*

## 3.3. Średni czas CPU przypadający na 1 wątek




*3.3. Wykres czasu CPU, przypadającego na pojedynczy wątek, w zależności od liczby wątków [ms]*

### 3.4. Liczba operacji na sekundę łącznego czasu CPU




*3.4. Wykres liczby operacji na sekundę czasu CPU w zależności od liczby wątków [1/s]*

## 4. Obserwacje

Możemy zaobserwować, że liczba wykonany operacji w czasie 10 sekund, na które były uruchamiane programy, jest znacznie większa w przypadku wykorzystania wzorca Active Object dla niewielkiej liczby wątków. Wraz ze wzrostem liczby wątków, liczba wykonanych operacji bardzo szybko spada, podczas gdy liczba wykonanych operacji w rozwiązaniu a 3 lockach pozostaje w przybliżeniu stała.

Rozwiązanie Active Object jest bardziej obciążające dla procesora, ponieważ oczekujące wątki wykonują w pętli obciążające procesor operacje, podczas gdy w rozwiązaniu na 3 lockach, są zatrzymywane na czas oczekiwania (nie wykonują wówczas obliczeń i czekają na odblokowanie).

Również liczba operacji, przypadające na pojedynczy wątek, jest początkowo znacznie większa w rozwiązaniu, wykorzystującym wzorzec Active Object. Przy zwiększaniu liczby wątków, liczba ta znacząco spada poniżej wartości otrzymywanych dla 3 locków.

## 5. Wnioski

Rozwiązanie korzystające ze wzorca Active Object jest lepsze, jeżeli nie chcemy blokować działania wątków w momencie, w którym oczekują one na zasoby. Podczas oczekiwania wątki mogą kontynuować działanie, wykonując inne zadania, w przeciwieństwie do 3 locków, gdzie wątki się wieszają, dopóki nie zostaną wznowione po pojawieniu się zasobów.

Aktywny obiekt pozwala również na bardziej efektywną obsługę niewielkiej liczby wątków niż rozwiązanie na 3 lockach.

Przy dużej liczbie wątków, gdy możemy sobie pozwolić na zatrzymywanie wątków czekających na zasoby, lepsze jest rozwiązanie wykorzystujące 3 locki.