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. Pomiary przy zmiennej pracy i stałej liczbie wątków

2.1. Opis eksperymentu

Zarówno implementacja problemu producenta-konsumenta z wykorzystaniem wzorca Active Object oraz implementacja z wykorzystaniem 3 locków zostały uruchomione dla 100-elementowego bufora oraz 4 producentów i 4 konsumentów. Podczas kolejnych testów zmieniana była wartość pracy (obciążenia) wątków oraz bufora/schedulera. Praca polegała na sumowaniu w pętli zadaną liczbę razy iloczynu sinusa i cosinusa wartości inkrementowanej w pętli.

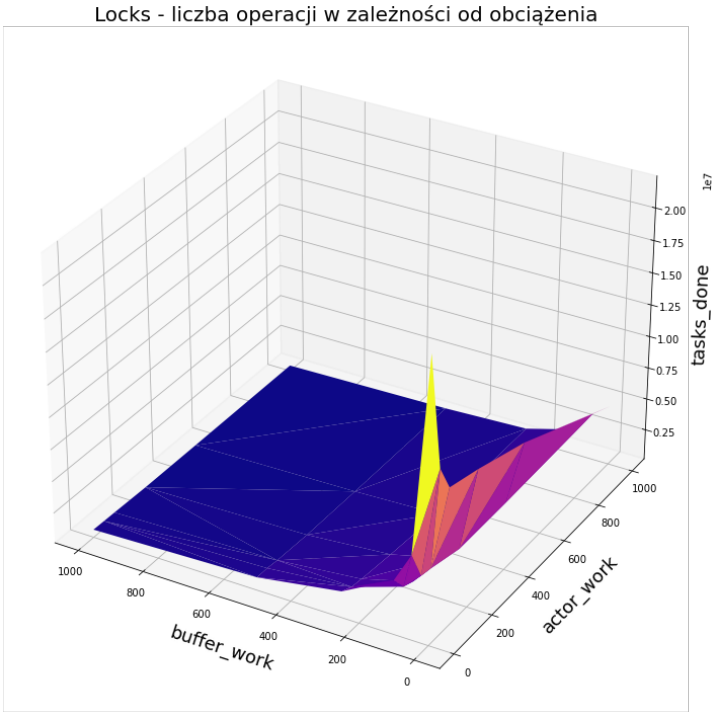
W tym eksperymencie mierzona była liczba wykonanych zadań w zależności od obciążenia dodatkową pracą.

Dla każdego zestawu parametrów (wielkości obciążenia bufora oraz wątków), oba programu były uruchamiane na czas 20 sekund, po czym wstrzymywana była

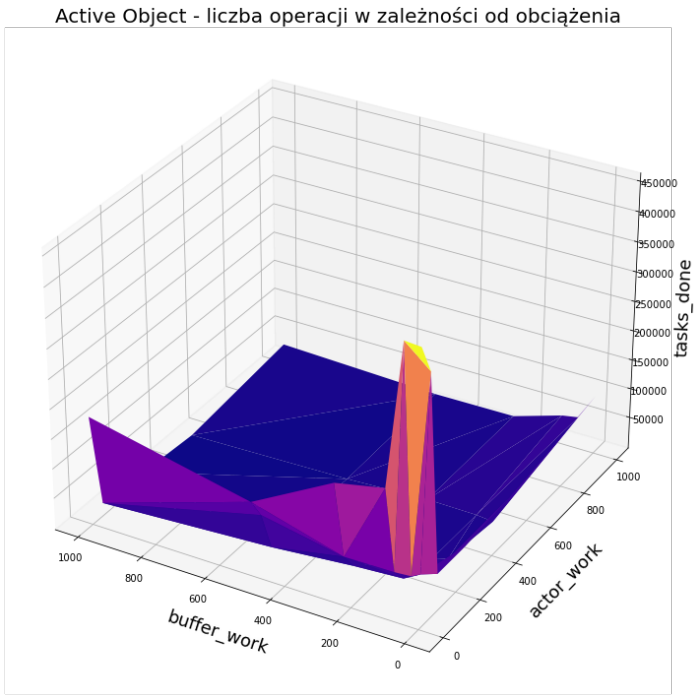
praca wątków i zapisywane wyniki. Każdy taki test był powtarzany 5 razy, a wyniki zostały uśrednione.

2.2. Wyniki eksperymentu

2.2.1. Liczba operacji w zależności od obciążenia



Rys. 2.1. Wykres liczby operacji w zależności od liczby wątków dla implementacji z wykorzystaniem locków



Rys. 2.2. Wykres liczby operacji w zależności od liczby wątków dla implementacji z wykorzystaniem wzorca active object

2.3. Obserwacje

Możemy zaobserwować, że w obu przypadkach, dodatkowe obciążenie zmniejsza liczbę wykonanych operacji. W przypadku rozwiązania na lockach, widzimy, że liczba ta spada dużo szybciej niż w przypadku wzorca Active Object. Dzieje się tak dlatego, że w rozwiązaniu na lockach, wątek, który przejmuje monitor, wykonując pracę na buforze, blokuje pozostałe wątki. W Active Object sytuacje jest trochę inna, ponieważ konsumenci oraz producenci mogą wciąż zgłaszać chęć produkcji/konsumpcji, podczas gdy scheduler jest zajęty, a więc działanie pozostałych wątków (poza wątkiem schedulera) nie jest wówczas blokowane.

Na rysunku 2.1. widać bardzo wyraźnie, że liczba wykonanych operacji spada dużo szybciej podczas zwiększania pracy bufora niż podczas zwiększania pracy konsumentów/producentów, co jest spowodowane wyżej opisanym faktem, iż dodatkowa praca bufora wstrzymuje działanie wszystkich wątków.

3. Pomiary przy stałej pracy i zmiennej liczbie wątków

3.1. Opis eksperymentu

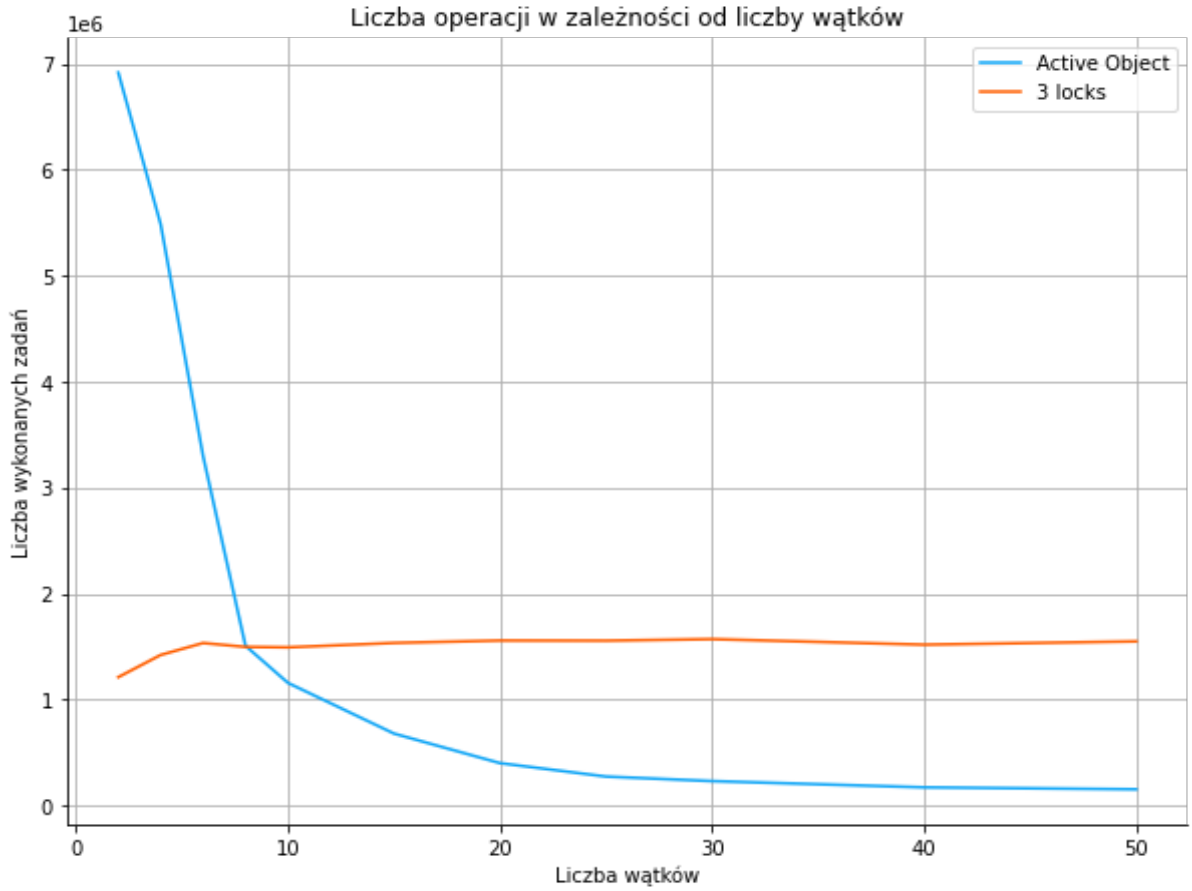
Zarówno implementacja problemu producenta-konsumenta z wykorzystaniem wzorca Active Object oraz implementacja w oparciu o 3 locki zostały uruchomione dla stałego czasu rzeczywistego, wynoszącego 10 sekund. Oba programy zostały uruchomione kolejno dla 2, 4, 6, 8, 10, 15, 20, 25, 30, 40 oraz 50 wątków[1]. 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.

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

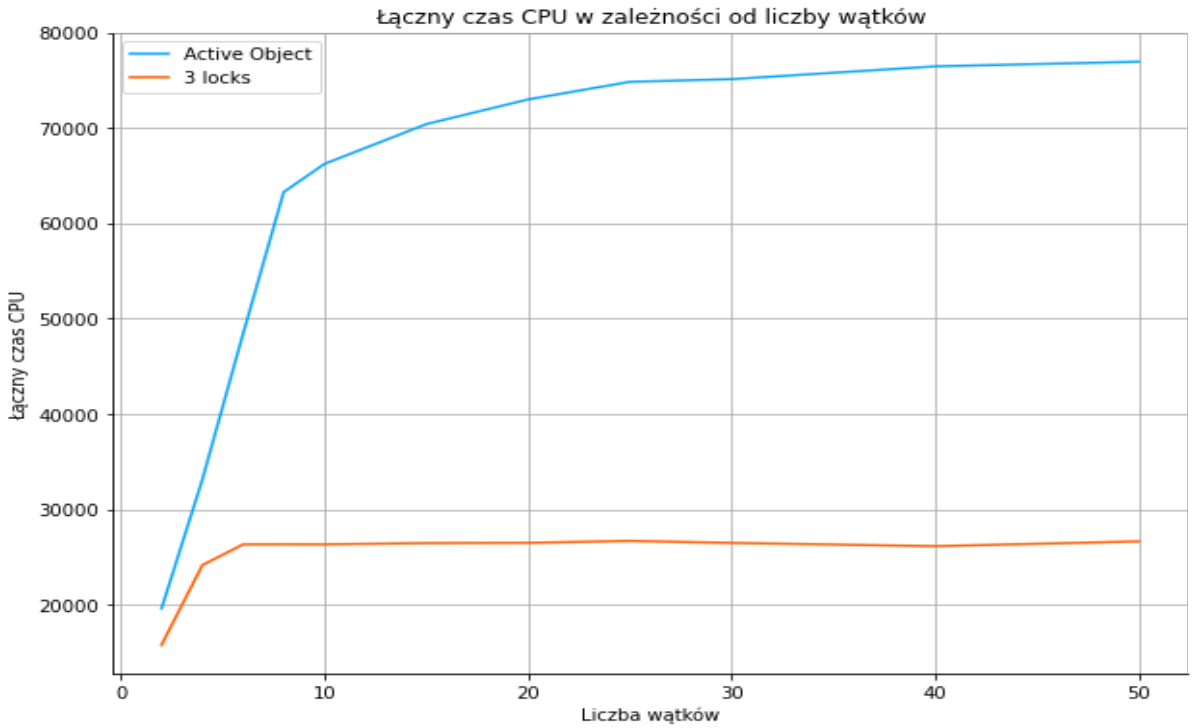
Po upływie 10 sekund czasu rzeczywistego, wszystkie wątki (producenci oraz konsumenci) zostały zatrzymane, po czym zmierzony został sumaryczny czas CPU, a także czas CPU przypadający na 1 wątek oraz liczba wykonanych operacji na buforze. Wyniki zostały zebrane w następnym punkcie sprawozdania.

[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.

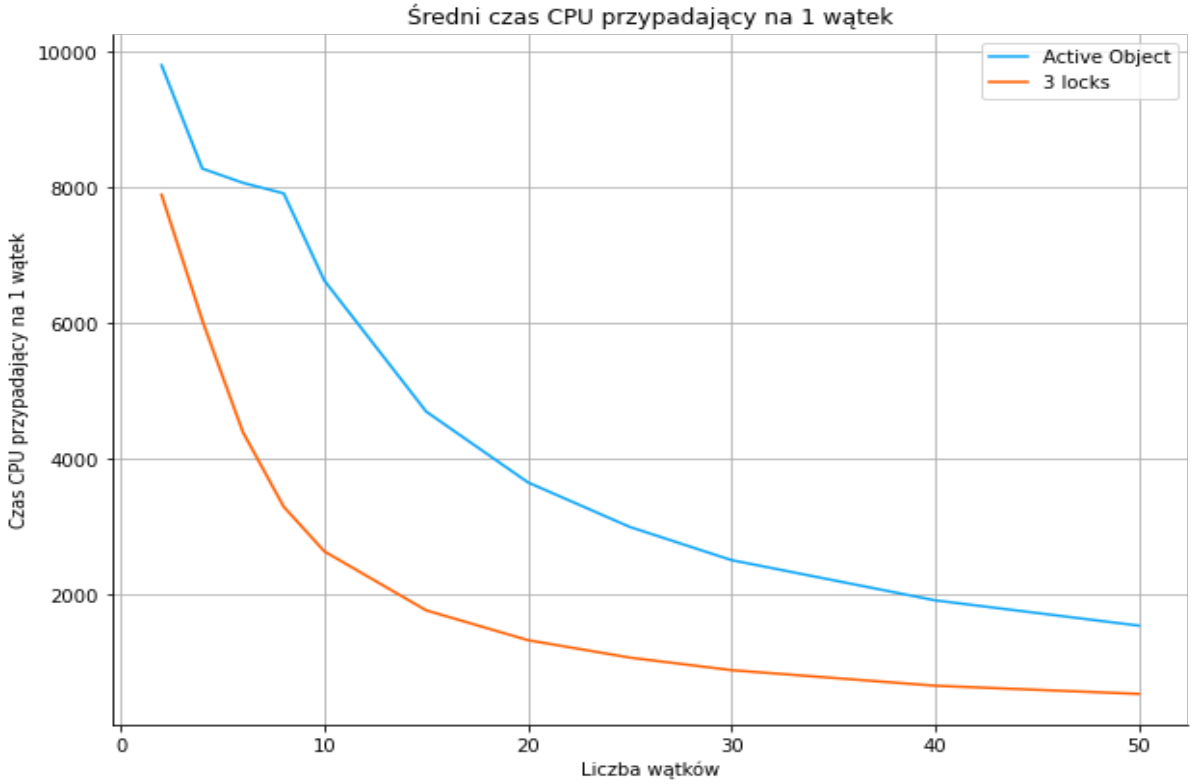
3.2. Wyniki eksperymentu



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



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



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

3.3. 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.

4. 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.