Teoria Współbieżności - Porównanie rozwiązań do problemu Producent-Konsument

Kacper Garus

13 listopada 2024

Spis treści

1 Wstęp

Celem ćwiczenia było porównanie ze sobą wydajności rozwiązań problemu Producent-Konsument z losową ilością zasobów, oraz porównanie rozwiązań nie zagładzających z rozwiązaniem zagładzającym.

2 Dane Techniczne

Komputer na którym zostały przeprowadzone obliczneia posiadał procesor Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz, mający 6 rdzeni i 12 procesorów logicznych. Działał on na systemie Windows 10 (64bit)

3 Przeprowadzone eksperymenty

3.1 Wydajność

Aby porównać wydajność dla każdego rozwiązania zliczyłem ilość operacji wykonanych w czasie 5 sekund.

3.2 Zagładzanie

Do porównania zagładzania dla każdego z rozwiązań zliczyłem ilość operacji wykonywanych przez każdy wątek, aby sprawdzić czy któryś z nich nie miał blokowanego dostępu do zasobów.

4 Implementacje

Zaprezentuję teraz swoje implementacje operacji dodawania elementów do bufora w każdym rozwiązaniu. Operacje Sub (odejmowanie elementów z bufora) wyglądają analogicznie. Cały kod został napisany w języku java.

4.1 Rozwiązanie z użyciem 2 conditions

```
public void Add(int howMany) throws InterruptedException {
    lock.lock();
    try {
        while (in_buf + howMany > bufferSize) {
            producerCond.await();
        }

        in_buf += howMany;

        consumerCond.signal();
    } finally {
        lock.unlock();
    }
}

public void Sub(int howMany) throws InterruptedException {
    lock.lock();
    try {
        while (in_buf - howMany < 0) {
            consumerCond.await();
        }

        in_buf -= howMany;

        producerCond.signal();
    } finally {
        lock.unlock();
    }
}
```

4.2 Rozwiązanie z użyciem 4 conditions

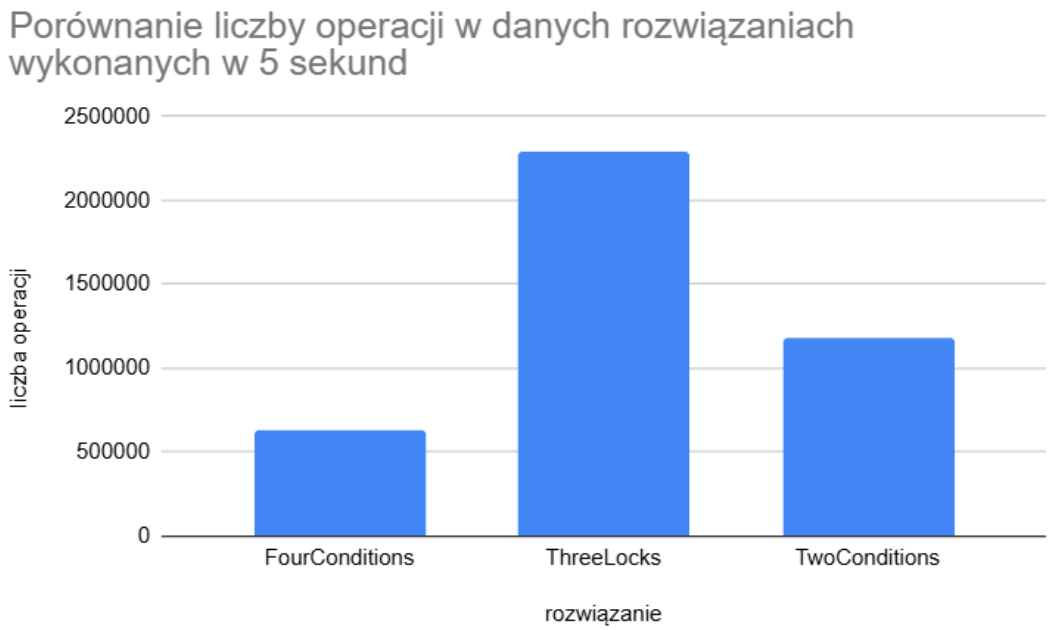
```
public void Add(int howMany) throws InterruptedException {
    lock.lock();
    try {
        while (isFirstProducerWaiting) {
            restProducerCond.await();
        }

        isFirstProducerWaiting = true;

        while (in_buf + howMany > bufferSize) {

            firstProducerCond.await();
        }

        in_buf += howMany;

        restProducerCond.signal();
        firstConsumerCond.signal();
        isFirstProducerWaiting = false;
    } finally {
        lock.unlock();
    }
}

public void Sub(int howMany) throws InterruptedException {
    lock.lock();
    try {
        while (isFirstConsumerWaiting) {
            restConsumerCond.await();
        }
        isFirstConsumerWaiting = true;

        while (in_buf - howMany < 0) {

            firstConsumerCond.await();
        }

        in_buf -= howMany;

        restConsumerCond.signal();
        firstProducerCond.signal();
        isFirstConsumerWaiting = false;
    } finally {
        lock.unlock();
    }
}
```

4.3 Rozwiązanie z użyciem 3 locks

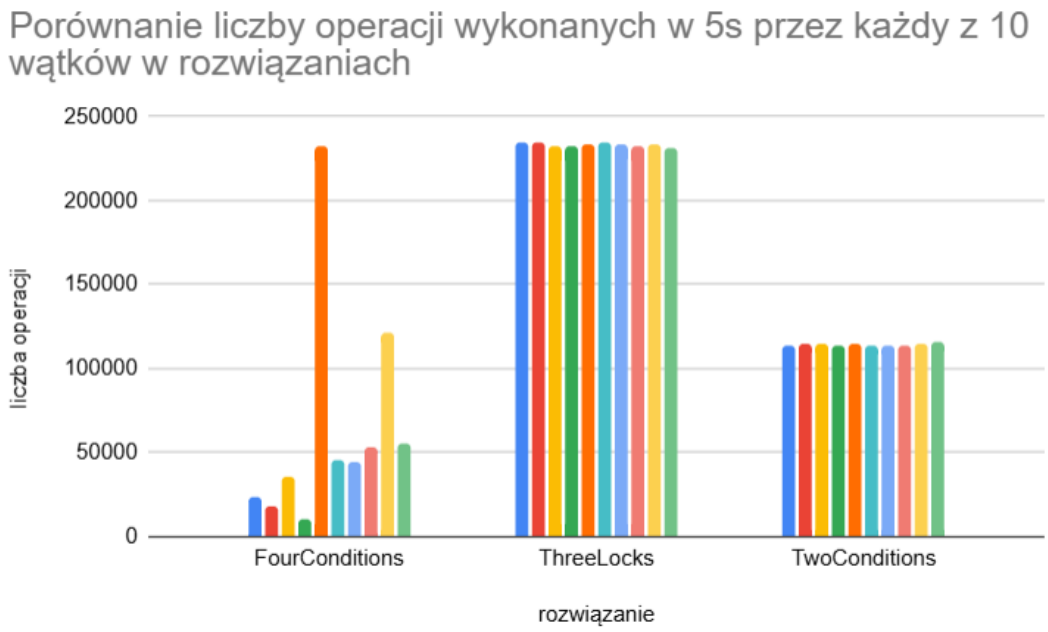
```
public void Add(int howMany) throws InterruptedException {
    lock1.lock();
    lock3.lock();
    try {
        while (in_buf + howMany > bufferSize) {
            queueWait.await();
        }

        in_buf += howMany;

        queueWait.signal();
    } finally {
        lock3.unlock();
        lock1.unlock();
    }
}

public void Sub(int howMany) throws InterruptedException {
    lock2.lock();
    lock3.lock();
    try {
        while (in_buf - howMany < 0) {
            queueWait.await();
        }

        in_buf -= howMany;

        queueWait.signal();
    } finally {
        lock3.unlock();
        lock2.unlock();
    }
}
```

5 Wyniki

5.1 Wykresy



Rysunek 1: Porównanie liczby wykonanych operacji w każdym rozwiązaniu w czasie 5 sekund.



Rysunek 2: Zestawienie ilości operacji wykonanych przez każdy wątek w każdym rozwiązaniu w czasie 5 sekund.

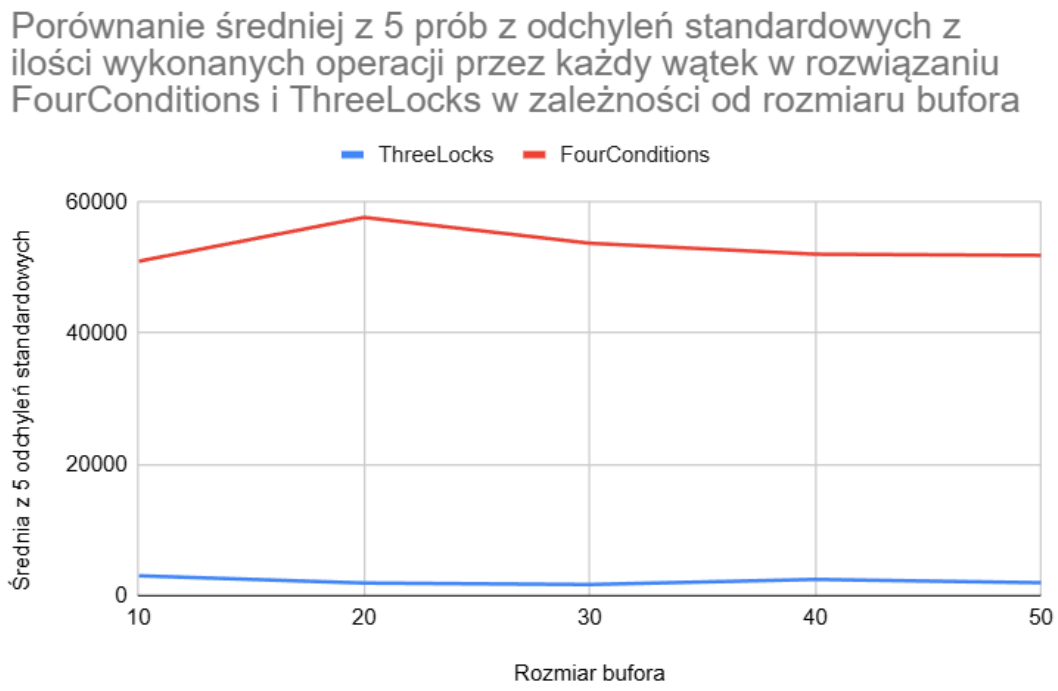
5.2 Wnioski na podstawie wykresów

Na podstawie wykresu 1 można zauważyć, że najwydajniejsze jest rozwiązanie z trzema lockami, które w danym czasie wykonuje ok. 4 razy więcej operacji od rozwiązania z czterema conditionami i ok. 2 razy więcej operacji od rozwiązania z dwoma conditionami. Wykres 2 który miał pokazać zagładzanie przy użyciu rozwiązania z dwoma conditionami pokazuje jednak zagładzanie w rozwiązaniu z czterema conditionami, co jest bardzo ciekawe i w teorii tak się nie powinno dziać.

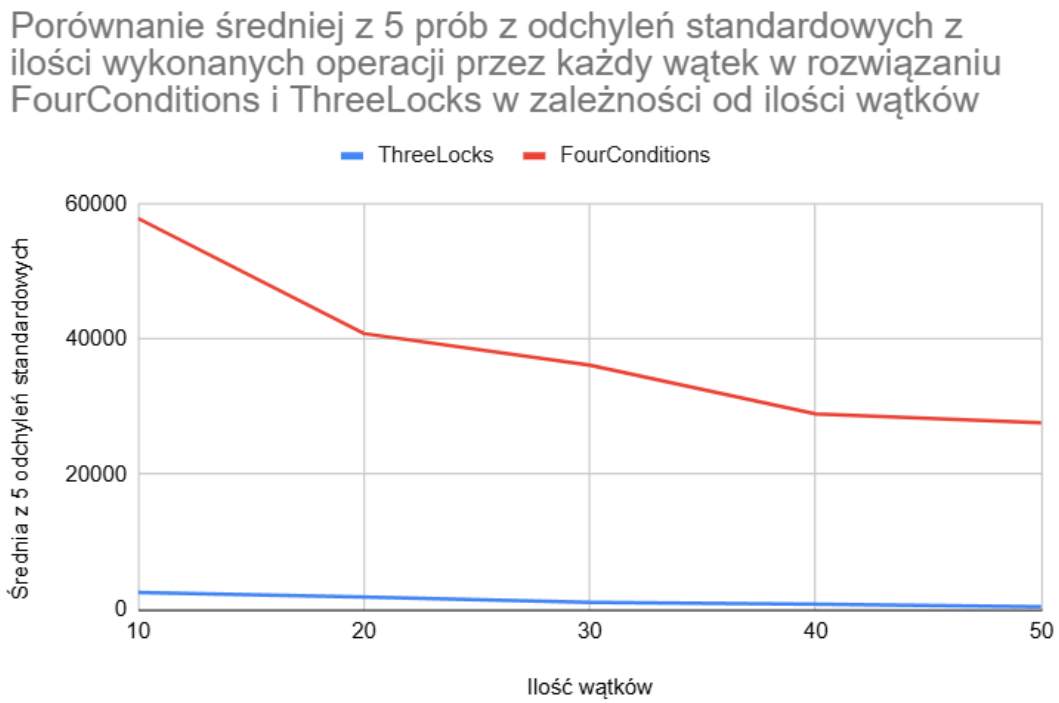
6 Problem głodzenia wątków w rozwiązaniu FourConditions

Na podstawie problemu zaobserwowanego na wykresie 2, sprawdzę jak się on zmienia w zależności od zmiany parametrów - rozmiaru bufora i ilości wątków. Wartością którą będę porównywał będzie średnia z pięciu prób z odchyleń standardowych z ilości operacji wykonanych przez każdy wątek w czasie pięciu sekund. Została ona wybrana dlatego, że dzięki niej najłatwiej będzie pokazać mi rozbieżność między ilościami operacji wykonanymi przez każdy wątek.

6.1 Wykresy



Rysunek 3: Porównanie średniej z odchyleń standardowych liczby wykonanych operacji przez każdy wątek w zależności od rozmiaru bufora dla rozwiązań FourConditions i ThreeLocks.



Rysunek 4: Porównanie średniej z odchyleń standardowych liczby wykonanych operacji przez każdy wątek w zależności od liczby wątków dla rozwiązań FourConditions i ThreeLocks..

6.2 Wnioski na podstawie wykresów

Na podstawie rysunku 3 mogę stwierdzić, że rozmiar bufora nie ma znaczenia, jeśli chodzi o głodzenie wątków. Natomiast na wykresie 4 widać, że ilośc wątków ma znaczenie w wartości badanego odchylenia standardowego, spada ono w obu rozwiązaniach i jest spodowane tym, że jeśli będziemy mieli więcej wątków to każdy z nich wykona proporcjonalnie mniej operacji, więc ich rozrzut będzie mniejszy.

7 Wnioski

Na podstawie przeprowadzonych eksperymentów mogę jednoznacznie stwierdzić, że najlepszym rozwiązaniem problemu Producent-Konsument będzie ThreeLocks, wykonuje ono najwięcej operacji w danym czasie, a wątki uzyskują równy dostęp do zasobów. Na nieoczekiwany problem związany z nierównym dostępem wątków do zasobów w rozwiązaniu FourConditions nie znalazłem żadnego rozsądnego wytłumaczenia, możliwe, że jest to związane z działaniem procesora lub języka Java.