Laboratorium VI – Teoria współbieżności

Weronika Szybińska, 28.11.2022

Treść zadania:

1. Problem czytelników i pisarzy proszę rozwiązać przy pomocy: semaforów i zmiennych warunkowych
2. Proszę wykonać pomiary dla różnej ilości czytelników (10-100) i pisarzy (od 1 do 10). W sprawozdaniu proszę narysować 3D wykres czasu w zależności od liczby wątków i go zinterpretować.
3. Proszę zaimplementować listę, w której każdy węzeł składa się z wartości typu Object, referencji do następnego węzła oraz zamka (lock).
4. Proszę porównać wydajność tego rozwiązania w stosunku do listy z jednym zamkiem blokującym dostęp do całości. Należy założyć, że koszt czasowy operacji na elemencie listy (porównanie, wstawianie obiektu) może być duży - proszę wykonać pomiary dla różnych wartości tego kosztu.

Opis rozwiązania:

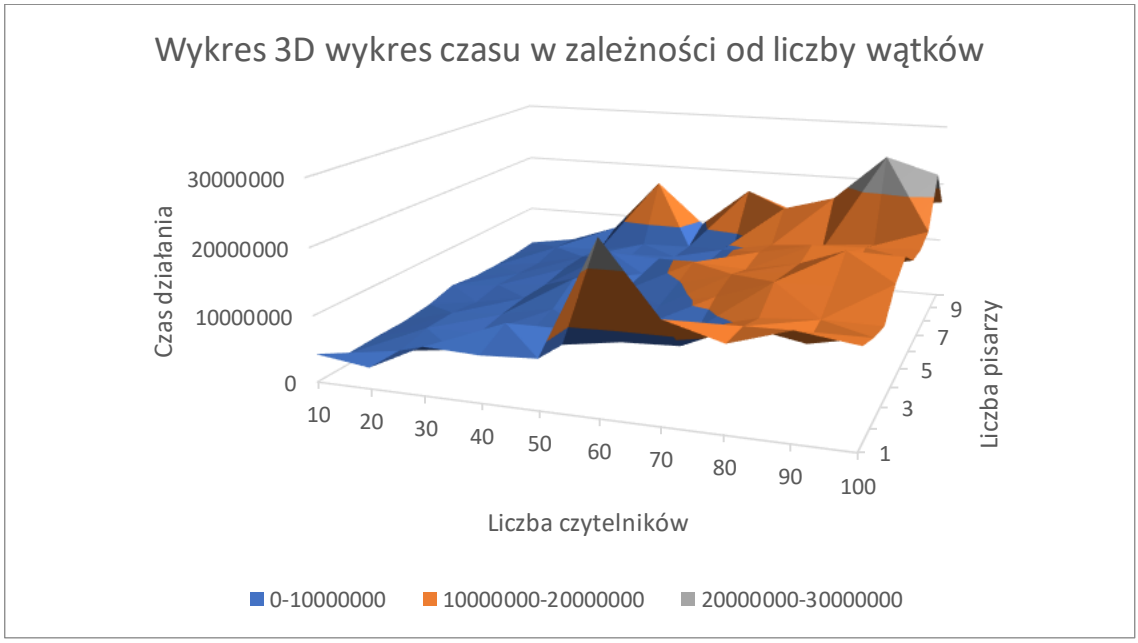
Do rozwiązania zadania pierwszego wykorzystana została klasa ReentrantLock oraz Semaphore. Na początku tworzonych jest N obiektów klasy Reader oraz M obiektów klasy Writer. Założeniami rozwiązania są: w księgarni może znajdować się wiele czytelników naraz, lecz tylko jeden pisarz. Do kontrolowania pisarzy wykorzystany został ReentrantLock, który pozwala przebywać w bibliotece tylko jednemu pisarzowi. Do kontroli reszty wątków wykorzystany został semafor. Pisarz, który aktualnie czeka na dostęp stopniowo zajmuje semafor, aż nie zablokuje go całego, wtedy blokuje możliwość dostępu do biblioteki czytelnikom. Po wyjściu z biblioteki odblokowuje cały semafor oraz ReentrantLock, dzięki czemu kolejny pisarz zaczyna próbę zablokowania czytelni. Poniżej przedstawiona została implementacja.

```java
class Reader extends Thread {
    private Library library;
    public Reader(Library library) {
        this.library = library;
    }
    @Override
    public void run() {
        try {
            library.Read();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```java
class Writer extends Thread {
    private Library library;
    public Writer(Library library) {
        this.library = library;
    }
    @Override
    public void run() {
        try {
            library.Write();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```java
class Library {
    private final Lock lock = new ReentrantLock();
    private final Semaphore semaphore ;
    private final int readerAmount;

    public Library(int readerAmount){
        this.readerAmount = readerAmount;
        semaphore = new Semaphore(readerAmount);
    }

    public void Read() throws InterruptedException {
        semaphore.acquire();
        System.out.println("read");
        semaphore.release();

    }

    public void Write() throws InterruptedException {
        lock.lock();
        try {
            for (int i = 0; i < readerAmount; i++) {
                semaphore.acquire();
            }
            System.out.println("write");
            semaphore.release(readerAmount);
        } finally {
            lock.unlock();
        }
    }
}
```

2. W zadaniu 2 wykonane zostały pomiaru czasu działania programów w zależności od ilości czytelników i pisarzy. Poniżej przedstawiony jest wykres 3D wykonany na ich podstawie.



3. W zadaniu trzecim zaimplementowana została lista zawierająca węzły zawierające zmienną typu Object, wskaźnik na kolejny węzeł oraz zamek. W liście zostały stworzone 3 metody: contains(Object o), remove(Object o) oraz add(Object o). Dostęp do węzłów jest zaimplementowany zgodnie z metodą blokowania drobnoziarnistego. Przechodząc przez listę przechowywane są w zmiennych wskaźniki do dwóch węzłów leżących obok siebie. Oba te węzły są na ten czas blokowane za pomocą zamków. Przy przechodzeniu do następnego węzła, dalszy węzeł jest odblokowywany a kolejny jest zablokowywany. Poniżej przedstawiona została implementacja programu.

```java
class FirstList {
    private Object value;
    private FirstList next;
    private Lock lock;

    public FirstList(Object val, FirstList next) {
        this.value = val;
        this.next = next;
        lock = new ReentrantLock();
    }

    public boolean contains(Object o) throws InterruptedException {
        FirstList prev = null;
        FirstList curr = this;
        curr.lock.lock();

        while (curr != null) {
            if (curr.value == o) {
                return true;
            }
            if (prev != null) {
                prev.lock.unlock();
            }
            prev = curr;
            prev.lock.lock();
            curr.lock.unlock();
            curr = curr.next;
            if (curr != null) {
                curr.lock.lock();
            }
        }
        return false;
    }
```

```java
    public boolean remove(Object o) throws InterruptedException {
        FirstList prev = null;
        FirstList curr = this;
        curr.lock.lock();
        if(curr.next != null){
            prev = curr;
            prev.lock.lock();
            curr.lock.unlock();
            curr = curr.next;
        }
        else if(curr.value == o){
            curr.value = null;
            return true;
        }
        else{
            return false;
        }
        while (curr.next != null) {
            if (prev.value == o) {
                prev.next = curr.next;
                return true;
            }
            prev = curr;
            prev.lock.lock();
            curr.lock.unlock();
            curr = curr.next;
            if (curr != null) {
                curr.lock.lock();
            }
        }
        if(curr.value == o){
            prev.next = null;
        }
        return false;
    }
```

```java
    public boolean add(Object o) throws InterruptedException {
        if (o == null) {
            return false;
        }
        FirstList curr = this;
        FirstList next = this.next;
        if(next == null){
            curr.next = new FirstList(o, next: null);
            return true;
        }
        curr.lock.lock();
        next.lock.lock();
        while (next != null) {
            curr.lock.unlock();
            curr = next;
            curr.lock.lock();
            next.lock.unlock();
            next = next.next;
            if(next != null) {
                next.lock.lock();
            }
        }
        curr.next = new FirstList(o,  next: null);
        return true;

    }

}
```

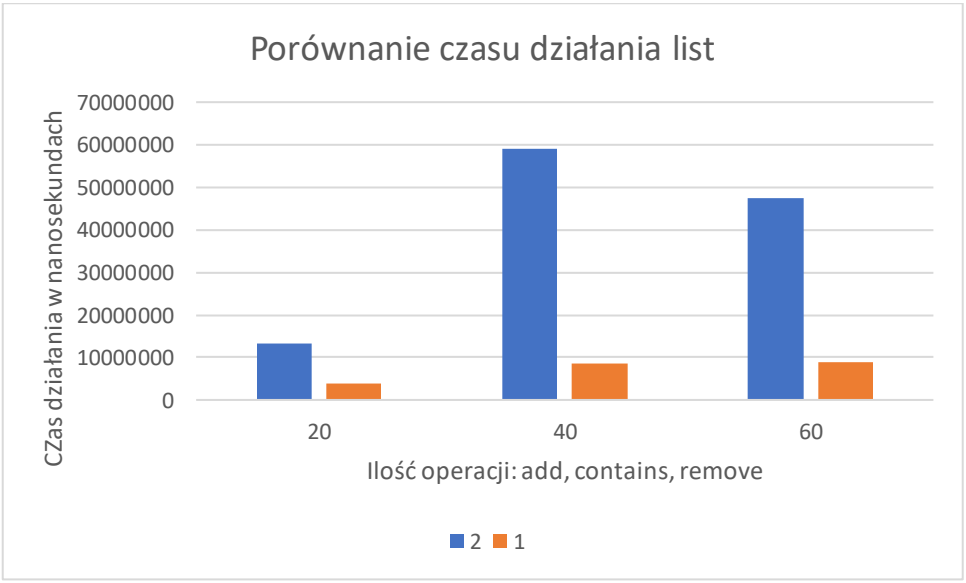
4.Do wykonania zadania czwartego stworzona została lista podobna do poprzedniej, lecz nieużywająca blokowania drobnoziarnistego a gruboziarnistego, tzn. jednego zamka na całą metodę. Poniżej przedstawiona została implementacja.

```
class SecondList {
    private Object val;
    private SecondList next;
    private static Lock lock = new ReentrantLock();
    private static long sleepTime;

    public SecondList(Object val, SecondList next) {
        this.val = val;
        this.next = next;
    }

    public boolean contains(Object o) throws InterruptedException {
        SecondList curr = this;
        lock.lock();
        try {
            while (curr != null) {
                if (val == o) {
                    return true;
                }
                curr = curr.next;
            }
        } finally {
            lock.unlock();
        }
        return false;
    }
```

```
    public boolean remove(Object o) throws InterruptedException {
        SecondList prev = null;
        SecondList curr = this;
        lock.lock();
        try {
            while (curr != null) {
                if (val == o) {
                    if (prev != null) {
                        prev.next = curr.next;
                        curr.next = null;
                    }
                    Thread.sleep( millis: sleepTime / 3);
                    return true;
                }
                prev = curr;
                curr = curr.next;
            }
        } finally {
            lock.unlock();
        }
        return false;
    }
```

```
    public boolean add(Object o) throws InterruptedException {
        if (o == null) {
            return false;
        }
        SecondList curr = this;
        SecondList next = this.next;
        lock.lock();
        try {
            while (next != null) {
                curr = next;
                next = next.next;
            }
            curr.next = new SecondList(o,  next: null);
            Thread.sleep(sleepTime);
            return true;
        } finally {
            lock.unlock();
        }
    }

}
```

Następnie zmierzone zostały czasy działania listy pierwszej oraz drugiej oraz stworzony został wykres porównujący otrzymane wyniki.



WNIOSKI:

W problemie czytelników i pisarzy dzięki wykorzystaniu Semafora do kontroli dostępu do biblioteki, ani pisarze ani czytelnicy nie są zagłodzeni, gdyż każdy pisarz dostanie w końcu dostęp do czytelni, zajmując stopniowo miejsce w semaforze. Dzięki wykorzystaniu zamków zapewnione jest, że w danym momencie tylko jeden pisarz próbuje zablokować semafor dzięki czemu nie występuje zjawisko blokady. Dzięki temu program dla różnych parametrów startowych ma bardzo podobne czasy działania. W zadaniu kolejnym widać, że wykorzystanie blokowania drobnoziarnistego znacznie skraca czas działania programu, gdyż daję możliwość dostępu do listy rożnym watkom w tym samym czasie.

BIBLIOGRAFIA:

https://home.agh.edu.pl/~funika/tw/lab6/

http://skinderowicz.pl/static/pw/wyklad6b.pdf