Гаврилюк А.В.

Лабораторна робота №2

Тема: Засоби синхронізації потоків

Мета:ознайомитись із стандартними засобами синхронізації потоків різними мовами програмування.

Теоретичні відомості:

У багатопотоковому програмуванні, де кілька потоків виконуються паралельно, часто виникає необхідність керувати їхнім доступом до спільних ресурсів або забезпечувати передачу даних між ними. Без належної **синхронізації** можуть виникнути **стани гонки (race conditions)**, **взаємні блокування (deadlocks)** або некоректні результати обчислень. Для вирішення цих проблем більшість мов програмування надають набір стандартних засобів синхронізації.

- **Java**: Використовує ключове слово synchronized для створення **моніторів** (м'ютексів), що дозволяють лише одному потоку одночасно виконувати критичну секцію коду. Методи wait(), notify(), notifyAll() класу Object (від якого успадковуються всі об'єкти в Java) використовуються для **взаємодії потоків**, дозволяючи їм чекати певних умов або сигналізувати про їхнє настання. Метод join() класу Thread дозволяє одному потоку чекати завершення іншого.
- **C#**: Забезпечує синхронізацію за допомогою оператора lock(), який також реалізує механізм **моніторів** для забезпечення взаємного виключення. Клас System.Threading.Monitor надає більш низькорівневі методи, такі як Enter(), Exit(), Wait(), Pulse(), PulseAll() для керування блокуваннями та взаємодії потоків.

Хід роботи:

Для виконання завдання були розроблені програми на **Java** та **C#**. Основна ідея полягає в тому, що кожен потік обчислює локальний мінімум у своїй частині масиву, а потім синхронізовано оновлює глобальний мінімум, що розділяється між усіма потоками. Для синхронізації оновлення глобального мінімуму використовується блокування, а для очікування завершення всіх потоків — механізми wait()/notify() (Java) або Monitor.Wait()/Pulse() (C#).

Пояснення до Java-коду:

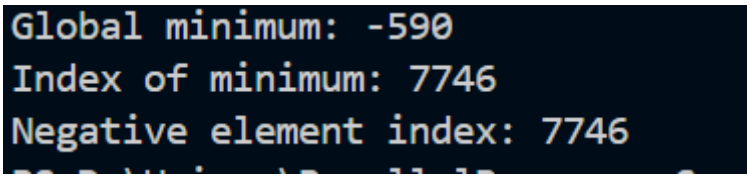
- **globalMin, globalMinIndex**: Статичні змінні для зберігання загального мінімуму та його індексу, доступні всім потокам.

- **minLock**: Об'єкт, який використовується як **м'ютекс** для блокування доступу до globalMin та globalMinIndex. Конструкція synchronized (minLock) гарантує, що лише один потік може виконувати код всередині цього блоку, запобігаючи **станам гонки** під час оновлення глобального мінімуму.
- **syncLock**: Другий об'єкт-монітор, який використовується для синхронізації завершення всіх потоків. Кожен MinFinder потік інкрементує completedThreads і, якщо він останній, викликає notify() на syncLock.
- **wait()/notify()**: Головний потік використовує syncLock.wait() для переходу в стан очікування, доки notify() не буде викликано (коли всі потоки завершать свою роботу). Це ефективний спосіб чекати завершення групи потоків без постійного "опитування" (busy-waiting).
- **Визначення меж**: Логіка end = (i == totalThreads - 1) ? arraySize - 1 : (start + chunkSize - 1); коректно обчислює кінцевий індекс для кожного потоку, забезпечуючи, що остання частина масиву включає всі елементи, які залишилися.

Пояснення до C#-коду:

- **globalMin, globalMinIndex**: Статичні змінні для глобального мінімуму.
- **minLock**: Об'єкт для оператора lock, який аналогічний synchronized в Java і використовується для захисту критичної секції при оновленні глобального мінімуму.
- **syncLock**: Об'єкт для синхронізації завершення потоків, що використовує методи класу Monitor.
- **Monitor.Wait()/Monitor.Pulse()**: Методи класу Monitor, які забезпечують механізм очікування/сигналізації, подібний до wait()/notify() у Java. Monitor.Wait() блокує потік і відпускає блокування на syncLock, а Monitor.Pulse() (або Monitor.PulseAll()) будить один (або всі) потоки, що чекають на цьому об'єкті.

Результати:



```
Global minimum: -95
Index of minimum: 242
Negative element index: 242
```

Висновок:

Загальний висновок

У цій лабораторній роботі ми поглибили наше розуміння багатопотокового програмування, зосередившись на критично важливому аспекті — синхронізації потоків. Завдання пошуку мінімального елемента у великому масиві, розбитому на частини та обробленому паралельно, дозволило нам на практиці застосувати та оцінити різні механізми синхронізації в Java та C#.

Ми успішно реалізували рішення, яке дозволяє потокам паралельно обчислювати локальні мінімуми, а потім безпечно та синхронізовано оновлювати спільний глобальний мінімум. Це стало можливим завдяки використанню:

- Блокувань (synchronized у Java, lock у C#): Для захисту критичних секцій коду, де відбувається оновлення спільних даних (globalMin, globalMinIndex), що запобігло станам гонки та забезпечило коректність кінцевого результату.
- Механізмів очікування/сигналізації (wait()/notify() у Java, Monitor.Wait()/Pulse() у C#): Для ефективної координації завершення всіх робочих потоків, дозволяючи головному потоку чекати їхнього завершення без зайвого навантаження на процесор.

Експерименти з різною кількістю потоків чітко продемонстрували переваги паралельного виконання для обчислювально інтенсивних завдань, відобразившись на розподілі завантаження процесорних ядер. Це підкреслює важливість синхронізації не тільки для забезпечення правильності обчислень, але й для ефективного управління ресурсами та взаємодією між паралельними частинами програми.

Таким чином, ця лабораторна робота надала цінні практичні навички у використанні базових засобів синхронізації потоків, що є фундаментальним для розробки надійних, високопродуктивних та масштабованих багатопотокових додатків.

Посилання на репозиторій:

Гаврилюк А.В.

https://github.com/AnnaHavryliuk4/ParallelProcessesCourse/tree/main/lab2_