Гаврилюк А.В.

Лабораторна робота №1

Тема: Засоби створення багатопоточних програм

Мета:Ознайомитись із засобами побудови паралельних програм різними мовами програмування.

Теоретичні відомості:

Сучасні програмні системи часто вимагають виконання кількох завдань одночасно для підвищення продуктивності, швидкодії та покращення взаємодії з користувачем. Для цього застосовується **багатопотоковість (multithreading)** – техніка, яка дозволяє програмі виконувати декілька частин коду, званих **потоками (threads)**, паралельно або псевдопаралельно (шляхом швидкого перемикання між ними на одному ядрі процесора).

Більшість популярних мов програмування надають вбудовані засоби або розширення бібліотек для роботи з багатопотоковістю. Наприклад, у **Java** базовим для розпаралелювання є клас Thread, тоді як у **C++** та **C#** додані аналогічні класи та бібліотеки (наприклад, std::thread у C++ та System.Threading.Thread у C#). Мова програмування **Ada** використовує специфічні програмні структури – **задачі (task)**.

Для керування доступом до спільних ресурсів та уникнення **станів гонки** (коли кілька потоків одночасно модифікують одні й ті ж дані, що може призвести до непередбачуваних результатів) використовуються **примітиви синхронізації**:

- **М'ютекси (Mutexes)**: Забезпечують взаємне виключення, дозволяючи лише одному потоку отримати доступ до критичної секції в будь-який момент часу.
- **Семафори (Semaphores)**: Керують доступом до обмеженої кількості ресурсів, дозволяючи заданій кількості потоків одночасно використовувати ресурс.
- **Монітори**: Механізми високого рівня, що об'єднують дані та методи для роботи з ними, забезпечуючи взаємне виключення автоматично.

Використання цих засобів дозволяє створювати ефективніші та більш чуйні програми, але вимагає ретельного проектування та синхронізації для уникнення типових проблем багатопотоковості, таких як стани гонки,

Хід роботи:

Для виконання завдання були розроблені програми на Java та C#. Основна логіка для обох мов подібна: створюється клас, який інкапсулює логіку

обчислення суми в окремому потоці, а головний потік керує їхнім запуском та завершенням.

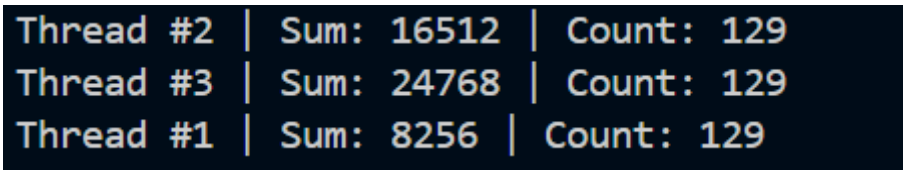
Пояснення до Java-коду:

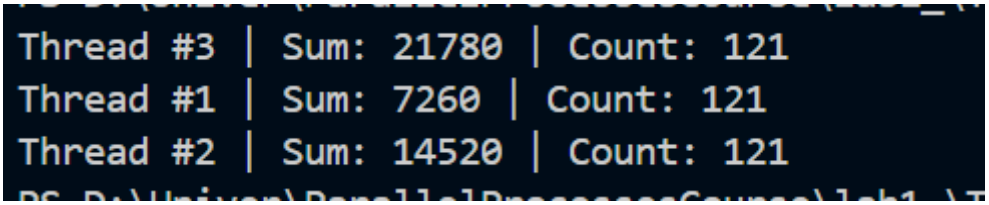
- **SumThread extends Thread**: Клас SumThread успадковує java.lang.Thread, що є одним зі стандартних способів створення потоків у Java. Метод run() містить логіку, яка виконуватиметься в окремому потоці.
- **volatile boolean running**: Використання ключового слова volatile для прапора running гарантує, що зміни цієї змінної, зроблені головним потоком, будуть негайно видимі для робочих потоків. Це запобігає проблемам кешування, які могли б призвести до того, що робочий потік не побачить зміну прапора.
- **stopThread()**: Метод для керування зупинкою потоку. Він встановлює running у false, що призводить до виходу з циклу while у методі run().
- **Thread.sleep(10)**: Імітує обчислювальну роботу та дозволяє планувальнику потоків перемикатися між потоками. Обробка InterruptedException важлива для коректної роботи з sleep.
- **Thread.sleep(2000) у main**: Головний потік робить паузу на 2 секунди, дозволяючи створеним потокам виконати обчислення.
- **threads[i].join()**: Після сигналу на зупинку, головний потік викликає join() для кожного робочого потоку. Це змушує головний потік чекати, доки відповідний робочий потік не завершить своє виконання, гарантуючи, що всі результати будуть виведені до завершення програми.

Пояснення до C#-коду:

- **SumThread**: Аналогічно Java-версії, містить логіку обчислення суми.
- **volatile bool running**: Використання volatile у C# також гарантує, що зміни прапора running будуть видимі в різних потоках.
- **Run() метод**: Містить основний цикл обчислень. Thread.Sleep(10) забезпечує призупинення потоку.
- **Stop() метод**: Встановлює прапор running у false.
- **new Thread(workers[i].Run)**: У C# конструктор Thread приймає делегат, що посилається на метод, який буде виконуватися в новому потоці.
- **Thread.Sleep(2000) та threads[i].Join()**: Функціональність та призначення цих методів аналогічні їхнім відповідникам у Java.

Результати:





Висновок:

У цій лабораторній роботі ми дослідили та застосували базові засоби створення багатопотокових програм. Працюючи з Java та C#, ми на практиці переконалися, як потоки (threads) дозволяють виконувати частини програми паралельно, підвищуючи її ефективність.

Основні результати, які ми спостерігали:

- Паралельне виконання: Створення кількох потоків для обчислення сум числових послідовностей продемонструвало, як задачі можуть виконуватися одночасно на різних ядрах процесора, що підтверджується моніторингом завантаження ЦП.
- Керування життєвим циклом потоків: Ми успішно реалізували механізм контрольованого запуску та зупинки потоків за допомогою прапора volatile та методів start() і join(). Це гарантує, що головний потік може ефективно керувати робочими потоками та очікувати їхнього завершення.
- Основи синхронізації: Хоча в цьому завданні не було явних станів гонки чи потреби в комплексних механізмах синхронізації (оскільки кожен потік працював зі своїми власними даними), розуміння volatile є першим кроком до безпечної взаємодії потоків.

Виконання цієї роботи допомогло глибше зрозуміти концепцію багатопотоковості, її переваги для підвищення продуктивності програм, а також важливість коректного керування потоками та їхньою синхронізацією. Отримані знання є фундаментальними для розробки складних, високопродуктивних та чуйних програмних систем.

Гаврилюк А.В.

Посилання на репозиторій:

https://github.com/AnnaHavryliuk4/ParallelProcessesCourse/tree/main/lab1_