Привіт! Я допоможу тобі створити дві версії програми на C#, які моделюють броунівський рух атомів домішок у одномірному кристалі з використанням мультипоточного програмування. Перша версія (Cells0) буде некоректною через відсутність блокування даних, а друга версія (Cells1) буде коректною із застосуванням механізму синхронізації.

**Підготовка середовища**

1. **Встановлення Visual Studio 2022 та .NET 8.0:**
   * Переконайся, що у тебе встановлено [Visual Studio 2022](https://visualstudio.microsoft.com/) з підтримкою .NET 8.0. Якщо ні, завантаж та встанови його з офіційного сайту.
2. **Створення двох окремих проектів:**
   * Відкрий Visual Studio 2022.
   * Обери **Create a new project**.
   * Вибери **Console App** (C#) і натисни **Next**.
   * Назви перший проект Cells0 і вибери відповідне розташування. Натисни **Create**.
   * Повтори ці кроки для другого проекту, назвавши його Cells1.

**Версія 1: Cells0 (Без блокування даних)**

**Опис**

Ця версія не використовує механізм синхронізації, тому доступ до спільного масиву cells може призвести до умов гонки (race conditions).

**Код**

csharp

Копировать код

// Файл: Program.cs в проекті Cells0

using System;

using System.Threading;

using System.Collections.Generic;

namespace Cells0

{

class Program

{

// Масив клітин кристалу

static int[] cells;

// Параметри моделювання

static int N; // Кількість клітин

static int K; // Кількість атомів домішок

static double p; // Порог вірогідності

// Список потоків

static List<Thread> threads = new List<Thread>();

// Флаг для зупинки потоків

static bool stop = false;

static void Main(string[] args)

{

// Встановлення параметрів

if (args.Length != 3)

{

Console.WriteLine("Використання: Cells0 N K p");

Console.WriteLine("Наприклад: Cells0 10 100 0.2");

return;

}

N = int.Parse(args[0]);

K = int.Parse(args[1]);

p = double.Parse(args[2]);

// Ініціалізація масиву клітин

cells = new int[N];

cells[0] = K; // Всі атоми спочатку в першій клітині

// Створення потоків

for (int i = 0; i < K; i++)

{

Thread t = new Thread(() => AtomMove());

threads.Add(t);

t.Start();

}

// Запуск знімків стану кожну секунду

Thread snapshotThread = new Thread(() => TakeSnapshots());

snapshotThread.Start();

// Моделювання 10 секунд

Thread.Sleep(10000);

stop = true;

// Очікування завершення потоків

foreach (var t in threads)

{

t.Join();

}

snapshotThread.Join();

// Підрахунок загальної кількості атомів

int total = 0;

for (int i = 0; i < N; i++)

{

total += cells[i];

}

Console.WriteLine($"Загальна кількість атомів домішки: {total}");

}

static void AtomMove()

{

Random rand = new Random(Thread.CurrentThread.ManagedThreadId + DateTime.Now.Millisecond);

int position = 0;

while (!stop)

{

double m = rand.NextDouble();

int newPos;

if (m > p)

{

newPos = position + 1;

}

else

{

newPos = position - 1;

}

// Віддзеркалення на краях

if (newPos < 0 || newPos >= N)

{

newPos = position;

}

// Оновлення масиву клітин

cells[position]--;

cells[newPos]++;

position = newPos;

// Коротка затримка для кращої симуляції

Thread.Sleep(1);

}

}

static void TakeSnapshots()

{

for (int i = 0; i < 10; i++)

{

Thread.Sleep(1000);

PrintCells(i + 1);

}

}

static void PrintCells(int second)

{

Console.WriteLine($"Стан клітин після {second} секунди:");

for (int i = 0; i < N; i++)

{

Console.Write($"{cells[i]} ");

}

Console.WriteLine();

}

}

}

**Пояснення коду**

1. **Параметри та ініціалізація:**
   * Програма приймає три параметри: N (кількість клітин), K (кількість атомів), p (порог вірогідності).
   * Ініціалізується масив cells розміром N, де всі атоми спочатку знаходяться в першій клітині.
2. **Потоки атомів:**
   * Для кожного атома створюється окремий потік, який виконує метод AtomMove.
   * Метод AtomMove генерує випадкове число m для визначення напрямку руху атома.
   * Відбувається оновлення позиції атома у масиві cells.
3. **Знімки стану:**
   * Окремий потік snapshotThread кожну секунду викликає метод PrintCells, який виводить поточний стан масиву cells.
4. **Зупинка потоків:**
   * Після 10 секунд програма встановлює прапорець stop = true для зупинки всіх потоків та очікує їх завершення.

**Запуск програми**

1. **Налаштування параметрів запуску:**
   * Відкрий властивості проекту Cells0 (правий клік на проекті в Solution Explorer → **Properties**).
   * Перейди до вкладки **Debug**.
   * У полі **Application arguments** введи: 10 100 0.2
2. **Запуск:**
   * Натисни **Start** або **Ctrl + F5** для запуску програми.
   * Програма буде працювати 10 секунд, виводячи стан клітин кожну секунду, а в кінці відобразить загальну кількість атомів.

**Проблеми версії Cells0**

* **Умови гонки (Race Conditions):** Оскільки декілька потоків одночасно модифікують спільний масив cells без блокування, можуть виникнути некоректні результати (наприклад, кількість атомів не буде збережена рівною K).
* **Непередбачувана поведінка:** Можлива нестабільність програми та різні результати при кожному запуску.

**Версія 2: Cells1 (З блокуванням даних)**

**Опис**

Ця версія використовує механізм синхронізації (lock) для забезпечення монопольного доступу до спільного масиву cells, тим самим запобігаючи умовам гонки.

**Код**

csharp

Копировать код

// Файл: Program.cs в проекті Cells1

using System;

using System.Threading;

using System.Collections.Generic;

namespace Cells1

{

class Program

{

// Масив клітин кристалу

static int[] cells;

// Параметри моделювання

static int N; // Кількість клітин

static int K; // Кількість атомів домішок

static double p; // Порог вірогідності

// Список потоків

static List<Thread> threads = new List<Thread>();

// Об'єкт для блокування

static object lockObj = new object();

// Флаг для зупинки потоків

static bool stop = false;

static void Main(string[] args)

{

// Встановлення параметрів

if (args.Length != 3)

{

Console.WriteLine("Використання: Cells1 N K p");

Console.WriteLine("Наприклад: Cells1 10 100 0.2");

return;

}

N = int.Parse(args[0]);

K = int.Parse(args[1]);

p = double.Parse(args[2]);

// Ініціалізація масиву клітин

cells = new int[N];

cells[0] = K; // Всі атоми спочатку в першій клітині

// Створення потоків

for (int i = 0; i < K; i++)

{

Thread t = new Thread(() => AtomMove());

threads.Add(t);

t.Start();

}

// Запуск знімків стану кожну секунду

Thread snapshotThread = new Thread(() => TakeSnapshots());

snapshotThread.Start();

// Моделювання 10 секунд

Thread.Sleep(10000);

stop = true;

// Очікування завершення потоків

foreach (var t in threads)

{

t.Join();

}

snapshotThread.Join();

// Підрахунок загальної кількості атомів

int total = 0;

for (int i = 0; i < N; i++)

{

total += cells[i];

}

Console.WriteLine($"Загальна кількість атомів домішки: {total}");

}

static void AtomMove()

{

Random rand = new Random(Thread.CurrentThread.ManagedThreadId + DateTime.Now.Millisecond);

int position = 0;

while (!stop)

{

double m = rand.NextDouble();

int newPos;

if (m > p)

{

newPos = position + 1;

}

else

{

newPos = position - 1;

}

// Віддзеркалення на краях

if (newPos < 0 || newPos >= N)

{

newPos = position;

}

// Блокування доступу до масиву клітин

lock (lockObj)

{

cells[position]--;

cells[newPos]++;

}

position = newPos;

// Коротка затримка для кращої симуляції

Thread.Sleep(1);

}

}

static void TakeSnapshots()

{

for (int i = 0; i < 10; i++)

{

Thread.Sleep(1000);

PrintCells(i + 1);

}

}

static void PrintCells(int second)

{

// Блокування доступу до масиву клітин перед читанням

lock (lockObj)

{

Console.WriteLine($"Стан клітин після {second} секунди:");

for (int i = 0; i < N; i++)

{

Console.Write($"{cells[i]} ");

}

Console.WriteLine();

}

}

// Публічний метод для отримання кількості атомів у клітинці i

public static int GetCell(int i)

{

lock (lockObj)

{

return cells[i];

}

}

}

}

**Пояснення коду**

1. **Механізм синхронізації:**
   * Використовується об'єкт lockObj для блокування доступу до спільного масиву cells.
   * У методах AtomMove та PrintCells обгорнуті блоки lock (lockObj) { ... }, що забезпечує монопольний доступ до масиву під час модифікації або читання.
2. **Загальні зміни:**
   * Структура програми схожа на Cells0, але додано блокування для коректної роботи з спільними даними.
3. **Публічний метод GetCell:**
   * Додано метод GetCell(int i), який дозволяє отримати кількість атомів у певній клітинці з безпечним доступом.

**Запуск програми**

1. **Налаштування параметрів запуску:**
   * Відкрий властивості проекту Cells1 (правий клік на проекті в Solution Explorer → **Properties**).
   * Перейди до вкладки **Debug**.
   * У полі **Application arguments** введи: 10 100 0.2
2. **Запуск:**
   * Натисни **Start** або **Ctrl + F5** для запуску програми.
   * Програма працюватиме 10 секунд, виводячи стан клітин кожну секунду, а в кінці відобразить загальну кількість атомів.

**Переваги версії Cells1**

* **Коректність:** Використання блокування запобігає умовам гонки, забезпечуючи правильне оновлення масиву cells.
* **Надійність:** Загальна кількість атомів домішки залишається незмінною (K), як і повинно бути за умовами задачі.

**Порівняння версій та дослідження впливу блокувань**

**Запуск та спостереження**

1. **Запусти обидві програми (Cells0 та Cells1) з однаковими параметрами:**
   * N = 10
   * K = 100
   * p = 0.2
2. **Спостереження результатів:**
   * У версії Cells0 ти можеш побачити, що загальна кількість атомів може відрізнятися від 100 через некоректні оновлення масиву.
   * У версії Cells1 загальна кількість атомів завжди залишиться 100.
3. **Аналіз швидкодії:**
   * Версія Cells0 може працювати швидше через відсутність блокувань, але її результати ненадійні.
   * Версія Cells1 буде мати невелике зниження швидкодії через блокування, але гарантує коректність результатів.

**Вплив гранулярності блокувань**

1. **Блокування всієї маси клітин (lockObj):**
   * Це найпростіший спосіб забезпечити синхронізацію, але він може призводити до зниження продуктивності при великій кількості потоків, оскільки всі атоми змагатимуться за монопольний доступ до масиву.
2. **Блокування окремих клітин:**
   * Для покращення продуктивності можна використовувати окремі блокування для кожної клітини. Це дозволить атомам, що працюють з різними клітинами, виконувати операції паралельно без блокування один одного.
   * Цей підхід вимагає складнішої реалізації, але може значно покращити швидкодію при великій кількості атомів та клітин.