Лабораторна робота №6
Розробка багатопотокових програм у C#

План
1. Поняття багатопотокової програми.
2. Клас Thread.
3. Завдання.

1. Паралелізмом називається одночасне виконання двох або більше операцій. У обчислювальних системах це означає, що комп'ютер виконує декілька незалежних операцій паралельно (одночасно), а не послідовно. Реальний паралелізм можливий тільки на комп'ютерах, обладнаних кількома процесорами або одним, але багатоядерним (слід зазначити, що деякі процесори з одним ядром мають кілька обчислювальних конвеєрів, що дає їм можливість паралельного виконання команд).

Потоком (thread) називається окремий шлях виконання програмного коду всередині програми (процесу), що виконується. Всі процеси в операційній системі Windows можуть містити кілька потоків (у такому випадку вони називаються **багатопотоковими)** . Головний потік процесу, який, наприклад, створюється при запуску методу Main() програми, написаної на C#, може створювати нові потоки, які паралельно виконують різні операції (рис. **Помилка! Джерело посилання не знайдено.**).

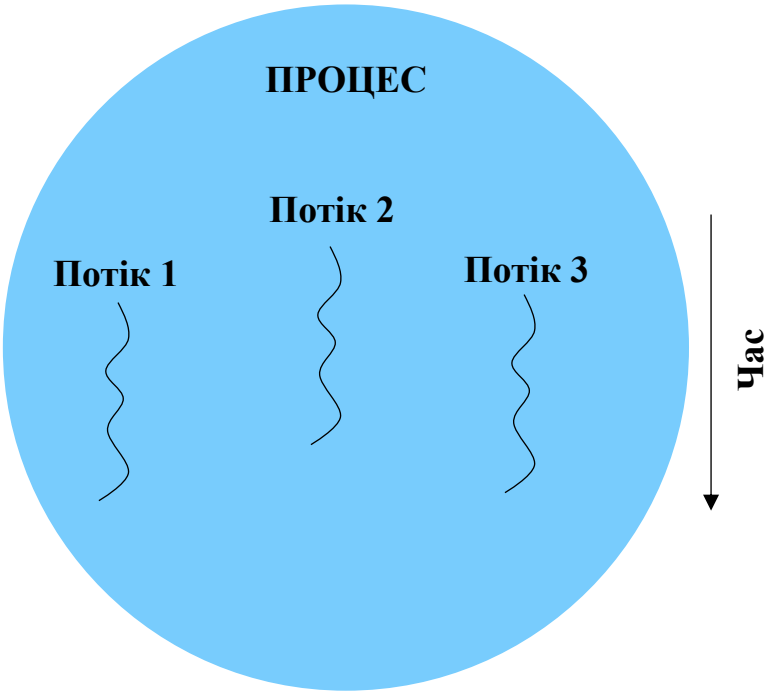

Рис. 1. Багатопотоковий процес

Класичним прикладом багатопотокового процесу у Windows є копіювання файлів, який у одному потоці здійснює безпосередньо копіювання, а в іншому – анімацію (тобто відображує віджет прогресу виконання операції) (рис. 2).

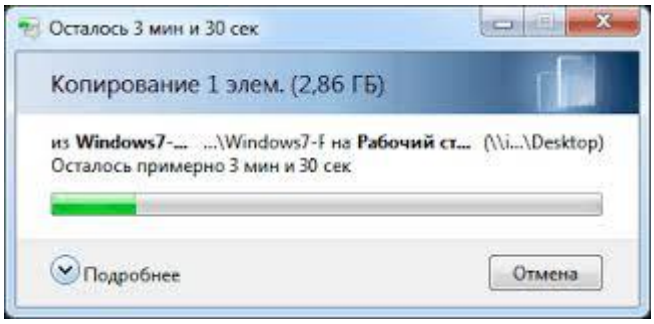

Рис. 1 – Приклад багатопотокового процесу у Windows

Створення багатопотокових програм дозволяє досягти реального паралелізму в роботі комп'ютера, обладнаного декількома процесорами (або одним багатоядерним процесором). Розробка багатопоточних застосунків має сенс навіть для однопроцесорних комп'ютерів, тому що використання багатопоточності дозволяє підвищити загальну реактивність системи за рахунок можливості обходу **блокувань** (застосунок, що очікує введення/виводу, блокується системою, але використання декількох потоків в ньому дозволяє реагувати на команди користувача у інших потоках).

Очевидно, що розробка багатопотокових застосунків практично є складнішою, ніж однопотокових. Програміст повинен правильно проектувати послідовність роботи потоків програми, щоб уникнути проблем, пов'язаних з так званими **гонитвами** (конкуренцією потоків за доступ до загальних ресурсів) і **клінчами** (взаємне блокування потоків при доступі до загальних ресурсів).

2. У платформі .NET для створення потоку використовується клас **Thread**, оголошений у просторі імен **System.Threading**. Основні його властивості та методи наведені у табл. 1

Таблиця 1. Основні члени класу Thread

Член класу	Призначення
CurrentThread	Посилання на поточний потік
IsAlive	Логічна ознака, чи запущено потік
IsBackground	Логічна ознака, чи потік є фоновим
Priority	Пріоритет потоку (за замовчуванням Normal – нормальний пріоритет)
ThreadState	Стан потоку (наприклад, Running – потік запущений, Stopped – потік було призупинено і т. п.)
Sleep()	Зупиняє поточний потік на задану кількість мілісекунд
Abort()	Припиняє виконання потоку, як тільки це буде можливо
Interrupt()	Зупиняє потік на заданий період очікування

Join()	Блокує батьківський потік до завершення зазначеного потоку
Resume()	Відновлює раніше зупинений потік
Start()	Запускає потік
Suspend()	Припиняє потік

Розглянемо наступний приклад.

```
using System;
using System.Threading;

namespace ConsoleThread
{
   class Program
   {
      public static int NumThread = 3; // Кількість потоків
      static void Main(string[] args)
      {
         Thread[] thr = new Thread[NumThread];

         // Запуск потоків
         for (int i = 0; i < NumThread; i++)
         {
            // Створення i-го потоку
            thr[i] = new Thread(new ThreadStart(DoWork));
            // Запуск потоку
            thr[i].Start();
         }
         // Очікування завершення потоків
         for (int i = 0; i < NumThread; i++)
            thr[i].Join();
         Console.ReadKey();
      }
      static void DoWork()
      {
         for (var i = 0; i < 5; i++)
         {
            Console.WriteLine("{0}", Thread.CurrentThread.GetHashCode());
            // Затримка на 10 мілісекунд (для наочності)
            Thread.Sleep(10);
         }
      }
   }
}
```

У цьому прикладі в циклі створюються три потоки, що паралельно виконують код методу DoWork(), в якому на екран у циклі виводиться хеш-код поточного потоку. Після створення та запуску потоків, головний потік блокується, щоб дочекатися завершення всіх дочірніх потоків. Якщо таке блокування не виконати, то можлива ситуація, коли головний потік завершиться раніше, ніж запущені ним потоки, що також призведе до їх завершення (навіть, якщо вони не закінчили своєї роботи).

Результат виконання вищенаведеної програми наведено на рис. 3.

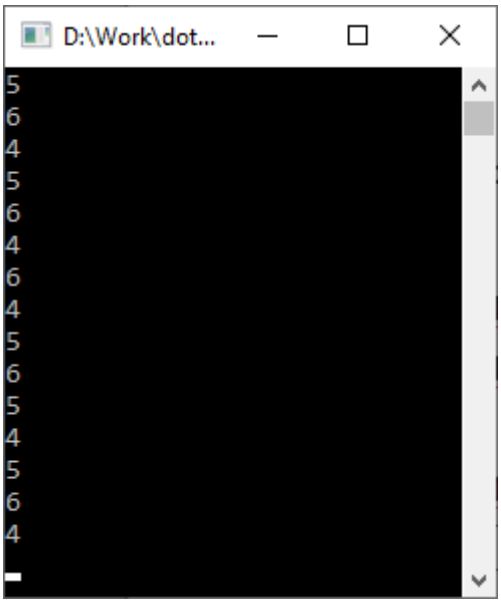

Рис. 2. Паралельна робота
трьох потоків

Для передачі параметрів потік можна скористатися підходом, наведеним нижче.

```
using System;
using System.Threading;

namespace ConsoleThread
{
   // Параметри потоку
   class TreadParams
   {
      public int begin, end; // Діапазон обчислюваних значень ряду в потоці
      public TreadParams(int b, int e)
      {
         begin = b;
         end = e;
      }
   }
   class Program
```

```
    {
        public static Mutex mutex = new Mutex(); // М’ютекс
        public static int NumThread = 6; // Кількість потоків
        public static double Sum = 0;    // Підсумкова сума
        static void Main(string[] args)
        {
            Thread[] thr = new Thread[NumThread];
            var MaxIter = 100000000; // Загальна кількість ітерацій
            var step = MaxIter / NumThread; // Кількість ітерацій у потоці

            // Запуск потоків
            for (int i = 0; i < NumThread; i++)
            {
                // Створення і-го потоку з можливістю приймати параметри
                thr[i] = new Thread(new ParameterizedThreadStart(CalcSeries));
                // Запуск потоку та передача в нього параметрів
                thr[i].Start(new TreadParams(i * step, (i == NumThread - 1) ?
                                                        MaxIter : (i + 1) * step));
            }
            // Очікування завершення потоків
            for (int i = 0; i < NumThread; i++)
                thr[i].Join();

            Console.WriteLine("Sum of series: {0}", Sum);
            Console.ReadKey();
        }
        // Обчислення суми ряду для заданого діапазону ітерацій
        public static void CalcSeries(object param)
        {
            double sum = 0;

            if (param is TreadParams)
            {
                for (double i = ((TreadParams)param).begin; i <
                                ((TreadParams)param).end; i++)
                    sum += (1.0 / (1 + i * i * i));
                mutex.WaitOne();
                Sum += sum;
                mutex.ReleaseMutex();
            }
        }
    }
}
```

У наведеній програмі розраховується значення суми ряду $\sum_{i=0}^{n} \frac{1}{1+i^3}$ для заданого параметра n. Ця сума розраховується у кількох потоках в такий спосіб, щоб кожен потік обчислював лише «власну» частину суми. Для передачі параметрів у потік використовується спеціально створений клас TreadParams, що містить дві властивості: початкове і кінцеве значення членів ряду, що обчислюються. Кількість значень ряду, що обчислюються, в потоці визначається співвідношенням step = MaxIter / NumThread, де MaxIter – загальна кількість обчислюваних членів ряду, а NumThread – кількість задіяних потоків. Тоді перший потік вважає члени ряду від 0 до step – 1; другий – від step до 2 * step – 1 і так далі. Останній потік обчислює суму членів ряду у діапазоні від (NumThread - 1) * step до MaxIter.

Тут також слід зазначити, що для попередження гонки (конкуренції за загальний ресурс – поле Sum класу Program) у цій програмі використовується м’ютекс. Він блокує доступ до Sum іншим потокам перед зміною цієї змінної в поточному потоці, після чого м'ютекс розблоковується, даючи можливість звернення до спільної змінної інших потоків.

Результат роботи програми наведено на рис. 4.

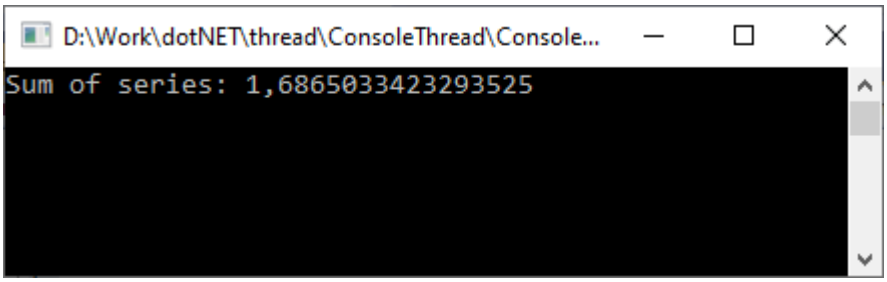

Рис. 3. Результат багатопотокового обчислення заданої кількості членів ряду

3. Написати багатопотокову програму обчислення заданої кількості членів ряду залежно від варіанта:

1 варіант – $\sum_{i=0}^{n} \frac{i}{1+i^4}$;

2 варіант – $\sum_{i=0}^{n} \frac{i^2}{1+i^4}$;

3 варіант – $\sum_{i=0}^{n} \frac{5}{i^2(1+i^2)}$;

4 варіант – $\sum_{i=0}^{n} \frac{i^{2/3}}{(1+i^4)^{1/2}}$;

5 варіант – $\sum_{i=1}^{n} \frac{i}{i}$;

Побудувати графік залежності часу роботи програми (див. Лекцію № 6) від кількості задіяних потоків.

Передбачити виведення на екран ПІБ та номери групи студента, який здає роботу.