Модуль 1. Багатопоточність

Лекція 2. Основи багатопоточності

Процеси і потоки. Класи для роботи з потоками .NET Framework. Клас Thread і його члени. Створення і запуск потоку. Призупинення потоку. Завершення потоку. Створення декількох потоків.

1. Процеси і потоки

У системах із спільною пам'яттю, включаючи багатоядерну архітектуру, паралельні обчислення можуть виконуватися як при виконанні багатьох процесів, так і при виконанні багатьох потоків. *Багатопроцесне виконання* має на увазі оформлення кожної підзадачі у вигляді окремої програми (процесу). Недоліком такого підходу є складність взаємодії підзадач. Кожен процес функціонує у своєму віртуальному адресному просторі, ізольованому від адресного простору іншого процесу. Для взаємодії підзадач необхідно використовувати спеціальні засоби міжпроцесної взаємодії (інтерфейси передачі повідомлень, спільні файли).

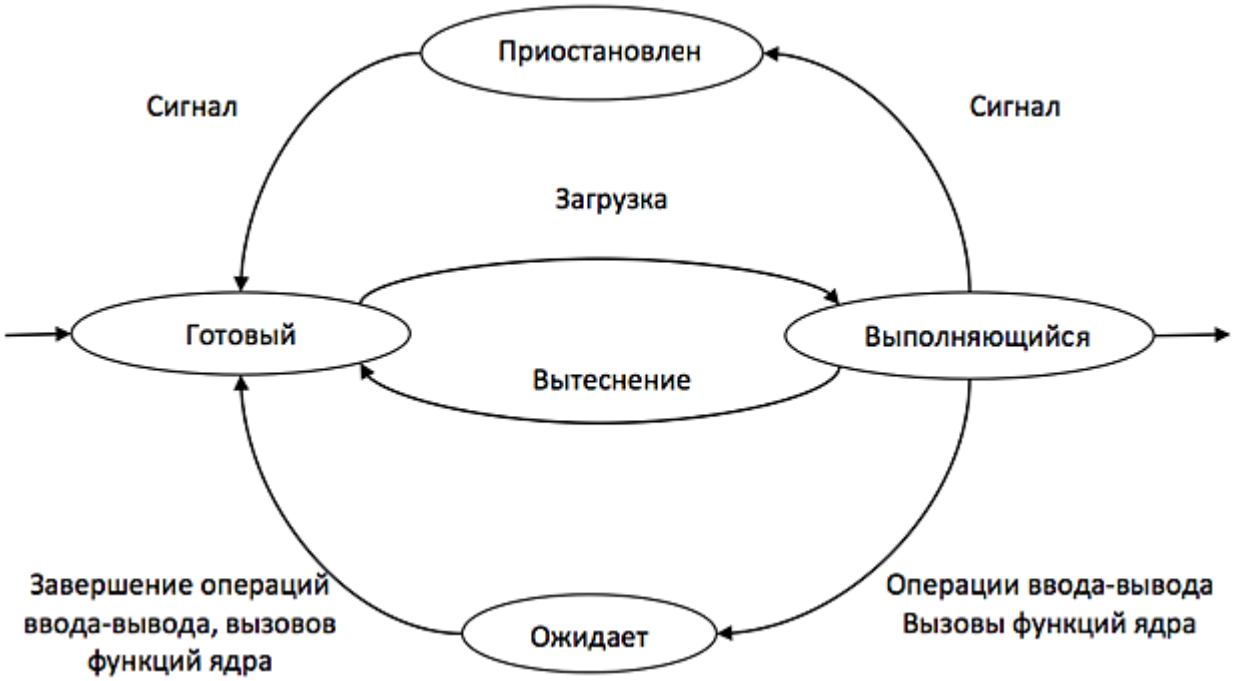
Потоки дозволяють виділити підзадачі у рамках одного процесу. Усі потоки одного застосунку працюють у рамках одного адресного процесу. Для взаємодії потоків не треба застосовувати які-небудь засоби взаємодії. Потоки можуть безпосередньо звертатися до спільних змінних, які змінюють інші потоки. Робота із спільними змінними призводить до необхідності використання засобів синхронізації, регулюючими порядок роботи потоків з даними.

1.1. Структура потоку

Потік складається з декількох структур. Ядро потоку - містить інформацію про поточний стан потоку: пріоритет потоку, програмний і стековий покажчики. Програмний і стекові покажчики утворюють контекст потоку і дозволяють відновити виконання потоку на процесорі. Блок оточення потоку - містить заголовок ланцюжка обробки виключень, локальне сховище даних для потоку і деякі структури даних, використовуваних інтерфейсом графічних пристроїв (GDI) і графікою OpenGL. Стек режиму користувача використовується для зберігання локальних змінних і аргументів, які передаються в методи. Стек режиму ядра - використовується, коли код програми передає аргументи у функцію операційної системи, що знаходиться в режимі ядра. Ядро ОС викликає власні методи і використовує стек режиму ядра для передачі локальних аргументів, а також для збереження локальних змінних.

1.2. Стани потоків

Кожний потік може знаходитися в одному з декількох *станів*. Потік, готовий до виконання і очікуючий надання доступу до центрального процесора, знаходиться в стані "**Готовий**". Потік, який виконується у поточний момент часу, має статус що "**Виконується**". При виконанні операцій введення-виведення або звернень до функцій ядра операційної системи, потік знімається з процесора і знаходиться в стані "**Чекає**". При завершенні операцій введення-виведення або поверненні з функцій ядра потік поміщається в чергу готових потоків. При перемиканні контексту потік вивантажується і поміщається в чергу готових потоків.



Перемикання контексту

1. Значення регістрів процесора для потоку, що виконується в даний момент, зберігаються в структурі контексту, яка розташовується в ядрі потоку.
2. З набору наявних потоків виділяється той, якому буде передано управління. Якщо вибраний потік належить іншому процесу, Windows перемикає для процесора віртуальний адресний простір.
3. Значення з вибраної структури контексту потоку завантажуються в регістри процесора.

Потоки в процесі розділяють спільно використовувані дані і мають власні стеки викликів і локальну пам'ять потоку (Thread Local Storage -TLS).

Головне – це те, що всі потоки виконуються в рамках **спільного адресного простору (в рамках одного процесу**).

2. Класи для роботи з потоками .NET Framework

Класи для роботи з потоками знаходяться у просторі імен System.Threading. У цьому просторі оголошуються типи, які використовуються для створення багатопотокових програм: робота з потоком, засоби синхронізації доступу до спільних даних, примітивний варіант класу Timer та інші.

```
using System.Threading;
```

Таблиця 1. Основні класи для роботи з потоками

Тип	Призначення
Interlocked	Синхронізація доступу до спільних даних
Monitor	Синхронізація потокових об'єктів за допомогою блокувань і управління чеканням
Mutex	Синхронізація ПРОЦЕСІВ
Thread	Власне клас потоку, що працює в середовищі виконання .NET. За допомогою цього класу створюються нові потоки.
ThreadPool	Клас, що надає засоби управління набором взаємозв'язаних потоків.
ThreadStart	Клас-делегат для методу, який має бути виконаний перед запуском потоку.
Timer	Варіант класу-делегата, який забезпечує передачу управління деякій функції-члену (неважливо якого класу!) у вказаний час. Сама процедура чекання виконується потоком в пулі потоків.
TimerCallback	клас-делегат для об'єктів класу Timer
WaitHandle	об'єкти-представники цього класу є об'єктами синхронізації (забезпечують багатократне чекання).
WaitCallback	Делегат, що представляє методи для робочих елементів (об'єктів) класу ThreadPool

Клас Thread. Загальна характеристика

Клас *Thread* представляє керовані потоки. Створює потоки і управляє ними: встановлює пріоритет і статус потоків. Цей клас містить властивості, статичні і нестатичні методи роботи з потоками.

Таблиця 2. Статичні члени класу Thread

Статичні члени класу Thread	Призначення
CurrentThread	Властивість. Лише для читання. Повертає посилання на потік, виконуваний в даний час (поточний потік).
GetData() SetData()	Обслуговування слоту поточного потоку.
GetDomain() GetDomainID()	Отримання посилання на домен застосування (на ID домена), в рамках якого працює вказаний потік.
Sleep()	Блокування виконання потоку на певний час (призупинення).

Таблиця 3. Нестатичні члени класу Thread

Нестатичні члени класу Thread	Призначення
IsAlive	Властивість. Якщо потік запущений, то true
IsBackground	Властивість. Робота у фоновому режимі.
Name	Властивість. Дружнє текстове ім'я потоку. Якщо потік ніяк не названий – значення властивості встановлене в null. Потік можна назвати 1 раз. Спроба перейменування потоку збуджує виключення. Область значень – значення перелічування (перечисление) ThreadState.
Priority	Властивість. Значення пріоритету потоку.
ThreadState	Властивість. Стан потоку. Область значень – значення перелічування ThreadState.
Interrupt()	Метод. Переривання роботи поточного потоку.
Join()	Метод. Чекання появи іншого потоку (або визначеного проміжку часу) з подальшим завершенням.
Resume()	Відновлення виконання потоку після призупинення.
Start()	Початок виконання раніше створеного потоку, представленого делегатом класу ThreadStart.
Suspend()	Призупинення виконання потоку.
Abort()	Завершення виконання потоку за допомогою генерації виключення – примусове завершення

3. Створення і запуск потоку

Потоки дозволяють розподіляти обчислення. Запустити потік можна єдиним способом – вказавши точку входу потоку - метод, до виконання операторів якого повинен приступити потік, що запускається.

В програмі один з потоків завжди є **головним** (або **первинним**). Він створюється автоматично при запуску програми. Інші потоки є **вторинними**.

Точкою входу **первинного** потоку є **статичний метод** Main або WinMain. Точніше, перший оператор методу.

Точка входу **вторинного** потоку призначається при створенні потоку.

Зауваження. Точкою входу в потік не може бути конструктор, оскільки не існує делегатів, які могли б налаштовуватися на конструктори.

Для створення потоку досить отримати екземпляр об'єкту типа Thread, тобто класу, визначеного в просторі імен System.Threading. Нижче наведена проста форма конструктора класу Thread:

public Thread(ThreadStart запуск)

де **запуск** — це ім'я методу, що викликається з метою почати виконання потоку, а ThreadStart — делегат, визначений в середовищі .NET Framework, як показано нижче.

public delegate void ThreadStart()

Отже, метод, що вказується як точка входу в потік, не повинен повертати результати і не приймати жодних аргументів.

Створений новий потік не почне виконуватися до тих пір, поки не буде викликаний його метод Start (), визначений в класі Thread. Існують дві форми оголошення методу Start (): *без параметрів і з параметрами*. Нижче наведена перша з них.

public void Start()

Раз почавшись, потік виконуватиметься до тих пір, поки не станеться повернення з методу, на який вказує запуск. Таким чином, після повернення з цього методу потік автоматично припиняється. Якщо ж спробувати викликати метод Start () для потоку, який вже почався, це призведе до генерування виключення ThreadStateException. У наведеному нижче прикладі програми створюється і починає виконуватися новий потік.

Приклад 1. Створення простої програми з двома потоками. Перший потік – головний запускає метод main, другий потік t1 створюється в main і запускає метод ThreadNew().

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Reflection;
using System.Diagnostics;

namespace Lab1_Thread1
{
    class Program
    {
        static void Main(string[] args)
        {
            Thread t1 = new Thread(ThreadNew);
            t1.Start();
            Console.WriteLine("Головий потік");
            Console.ReadKey();
        }
        static void ThreadNew()
        {
            Console.WriteLine("Другий потік");
        }
    }
}
```

В цій програмі у якості точки входу другого потоку викликається *статичний метод* ThreadNew().

Але як і в однопоточній програмі, методи, які виконуються вторинними потоками, можуть бути членами інших класів.

Приклад 2.

В цьому прикладі є клас MyThread, що містить метод Run(), який викликається у вторинному потоці.

В програмі спочатку визначається клас MyThread, призначений для створення другого потоку. У методі Run () цього класу організовується цикл для підрахунку від 0 до 9.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Reflection;
using System.Diagnostics;

namespace Lab1_Thread1
{
    class MyThread
    {
        public int count;
        string thrdName;
        public MyThread(string name)
        {
            count = 0;
            thrdName = name;
        }
        // Точка входу в потік,
        public void Run() {
          Console.WriteLine(thrdName + " почався");
            do {
                Thread.Sleep(500);
                Console.WriteLine("В потоці " + thrdName + ", count = " + count);
                count++;
           } while(count < 10) ;
         Console.WriteLine(thrdName + " закінчився");
       }
  }
    class MultiThread
    {
        static void Main() {
        Console.WriteLine("Головний потік почався");
        // Спочатку створити об'єкт класу MyThread.
        MyThread mt = new MyThread("Нащадок #1");
        // Потім створити потік з цього об'єкту.
        Thread newThrd = new Thread(mt.Run);
       // запустити потік на виконання.
        newThrd.Start();
        do {
        Console.WriteLine("Працює Головний потік.");
       // призупинити головний потік на 500 мс для завершення виконання другого потоку
        Thread.Sleep(500);
        } while (mt.count != 10);
        Console.WriteLine("Головний потік закінчився");
        Console.ReadKey();
      }
  }

}
```

Потік створюється за рахунок виклику методу Start() об'єкту Thread. Проте після виклику методу Start() новий потік все ще перебуває у стані Unstarted. У стан Running потік переходить відразу після того, як планувальник

потоків операційної системи вибере його для виконання. Інформація про поточний стан потоку доступна через властивість Thread.ThreadState.

Примітка! Головний потік нічим не краще за будь-які інші потоки програми. Він може раптово завершитися раніше всіх ним же створених потоків! Програма завершується після виконання ОСТАННЬОЇ команди в ОСТАННЬОМУ виконуваному потоці. Неважливо в якому.

Зверніть увагу на виклик статичного методу **Sleep (500)**, визначеного в класі Thread. Цей метод обумовлює припинення того потоку, з якого він був викликаний на певний період часу, що вказується в мілісекундах. Коли призупиняється один потік, може виконуватися інший. У цій програмі використовується наступна форма методу Sleep ():

public static void Sleep(int мілісекунд_простою)

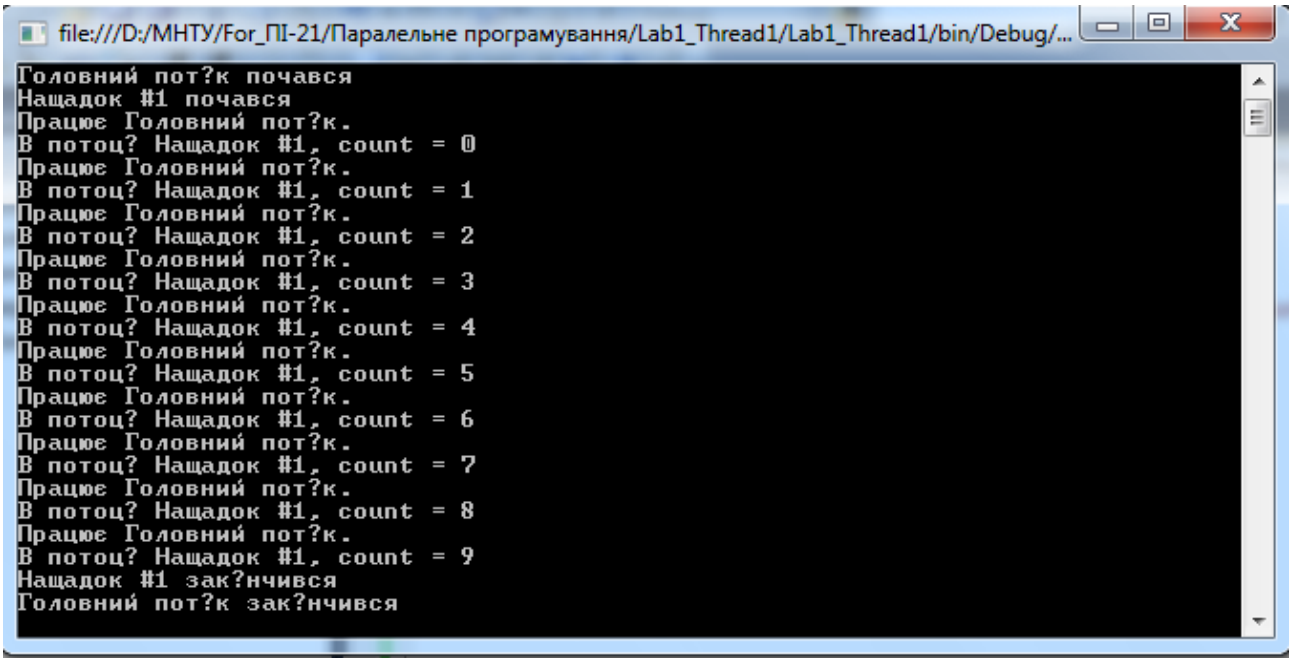
де мілісекунд_простою позначає період часу, на який припиняється виконання потоку. Якщо вказана кількість мілісекунд_простою рівна нулю, то потік призупиняється лише для того, щоб надати можливість для виконання потоку, який чекає своєї черги.

У методі Main () новий об'єкт типу Thread створюється за допомогою наведеної нижче послідовності операторів.

```
// Спочатку сконструювати об'єкт типа MyThread.
MyThread mt = new MyThread("Нащадок #1");
// Далі сконструювати потік з цього об'єкту.
Thread newThrd = new Thread(mt.Run);
// І нарешті, почати виконання потоку.
newThrd.Start();
```

Як видно з коментарів до наведеного вище фрагмента коду, спочатку створюється об'єкт типу MyThread. Потім цей об'єкт використовується для створення об'єкту типа Thread, для чого конструктору цього об'єкту як точка входу передається метод **mt. Run ().** І нарешті, виконання потоку починається з виклику методу Start ().

Завдяки цьому метод mt.Run () виконується в своєму власному потоці. Після виклику методу Start () виконання основного потоку повертається до методу Main (), де починається цикл do-while. Обидва потоки продовжують виконуватися, спільно використовуючи ЦП, аж до закінчення циклу. Нижче наведений результат виконання програми. (Він може відрізнятися залежно від середовища виконання операційної системи і завантаження завдань.)



Часто в багатопоточній програмі потрібно, щоб основний потік був останнім потоком, що завершує її виконання. Формально програма продовжує виконуватися до тих пір, поки не завершаться всі її пріоритетні потоки. Тому вимагати, щоб основний потік завершував виконання програми, зовсім не обов'язково. Проте цього правила прийнято дотримуватися в багатопоточному програмуванні, оскільки воно явно визначає кінцеву точку програми.

У розглянутій вище програмі зроблена спроба зробити основний потік завершуючим виконання. Для цієї мети значення змінної count перевіряється в циклі do-while усередині методу Main (), і як тільки це значення виявляється рівним 10, цикл завершується і відбувається почергове повернення з методів Sleep (). Але такий підхід далекий від досконалості, тому розглянемо як можна вдосконалити цю програму.

Прості способи удосконалення багатопотокової програми

Розглянута вище програма працездатна, але її можна зробити ефективнішою, якщо внести в неї ряд простих удосконалень.

По-перше, можна зробити так, щоб виконання потоку починалося відразу ж після його створення. Для цього досить отримати екземпляр об'єкту типа Thread в конструкторі класу MyThread.

По-друге, в класі MyThread зовсім не обов'язково зберігати ім'я потоку, оскільки для цього в класі Thread спеціально визначена властивість Name.

public string Name { get; set; }

Властивість Name доступна для запису і читання і тому може служити як для запам'ятовування, так і для читання імені потоку.

Нижче наведена версія попередньої програми, до якої внесені згадані вище удосконалення.

Приклад 3.

```
// Інший спосіб запуску потоку.
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Reflection;
using System.Diagnostics;

namespace Lab1_Thread2
{
    class MyThread
    {
        public int count;
        public Thread Thrd;
        public MyThread(string name)
        {
            count = 0;
            Thrd = new Thread(this.Run);
            Thrd.Name = name; // задати ім'я потоку
            Thrd.Start(); // почати потік
        }
        // Точка входу в потік,
        public void Run() {
            Console.WriteLine(Thrd.Name + " почався");
            do {
                Thread.Sleep(500);
                Console.WriteLine("В потоці " + Thrd.Name + ", count = " + count);
                count++;
           } while(count < 10) ;
            Console.WriteLine(Thrd.Name + " закінчився");
       }
  }
    class MultiThread
    {
        static void Main() {
        Console.WriteLine("Головний потік почався");
        // Спочатку створит об'єкт класу MyThread.
        MyThread mt = new MyThread("Нащадок #1");
        do {
        Console.WriteLine("Працює Головний потік.");
       // призупинити головний потік на 500 мс для завершення виконання другого потоку
         Thread.Sleep(100);
        } while (mt.count != 10);
        Console.WriteLine("Головний потік закінчився");
        Console.ReadKey();
      }
  }

}
```

Ця версія програми дає такий самий результат, як і попередня. Зверніть увагу на те, що об'єкт потоку зберігається в змінній Thrd з класу MyThread.

4. Призупинення потоків

Призупинення виконання потоку забезпечується статичним методом **Sleep(),** який припиняє виконання того потоку, **з якого** він був викликаний на певний період часу, що вказується в мілісекундах. Коли призупиняється один потік, може виконуватися інший.

Виконання методів поточного потоку блокується на певні інтервали часу. Все залежить від вибору перегруженого варіанту методу. Планувальник потоків

дивиться на потік і приймає рішення відносно того, чи можна продовжити виконання призупиненого потоку. У найпростішому випадку цілочисельний параметр визначає часовий інтервал блокування потоку в мілісекундах.

Якщо значення параметра встановлене в 0, потік буде зупинений до того моменту, поки не буде наданий черговий інтервал для виконання операторів потоку.

Якщо значення інтервалу задане за допомогою об'єкту класу TimeSpan, то момент коли може бути відновлене виконання потоку, визначається з урахуванням закодованої в цьому об'єкті інформації.

// Потік зупинено на 1 годину, 2 хвилини, 3 секунди:

Thread.Sleep(new TimeSpan(1,2,3));
:::::

// Потік зупинено на 1 день, 2 години, 3 хвилини, 4 секунди, 5 мілісекунд:

Thread.Sleep(new TimeSpan(1,2,3,4,5));

Значення параметра, представлене виразом

System.Threading.Timeout.Infinite
дозволяє призупинити потік на невизначений час. А розбудити потік при цьому можна за допомогою методу **Interrupt(),** який в цьому випадку викликається з іншого потоку.

5. Іменування потоку

При створенні, потік не має імені (властивість Name встановлена в null. Іменований потік не можна перейменувати, це призведе до виключення.

Приклад 4.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Diagnostics;

namespace Ex1_Thread
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 0;
            bool isNamed = false;
            do
                {
                try
                    {
                    if (Thread.CurrentThread.Name == null)
                        {
                        Console.Write("Get the name for current Thread > ");
                        Thread.CurrentThread.Name = Console.ReadLine();
```

```
                    }
                else
                    {
                    Console.WriteLine("Current Thread : {0}.",
Thread.CurrentThread.Name);
                    if (!isNamed)
                        {
                        Console.Write("Rename it. Please...");
                        Thread.CurrentThread.Name = Console.ReadLine();
                        }
                    }
                    }
                catch (InvalidOperationException e)
                {
                    Console.WriteLine("{0}:{1}",e,e.Message);
                    isNamed = true;
                }
            i++;
            }
        while (i < 10);
        Console.ReadKey();
    }
  }
  }
```

5. Створення декількох потоків

У попередніх прикладах програм був створений лише один вторинний потік. Але в програмі можна створити стільки потоків, скільки буде потрібно. Наприклад, у наступній програмі створюються три потоки.

Приклад 5.

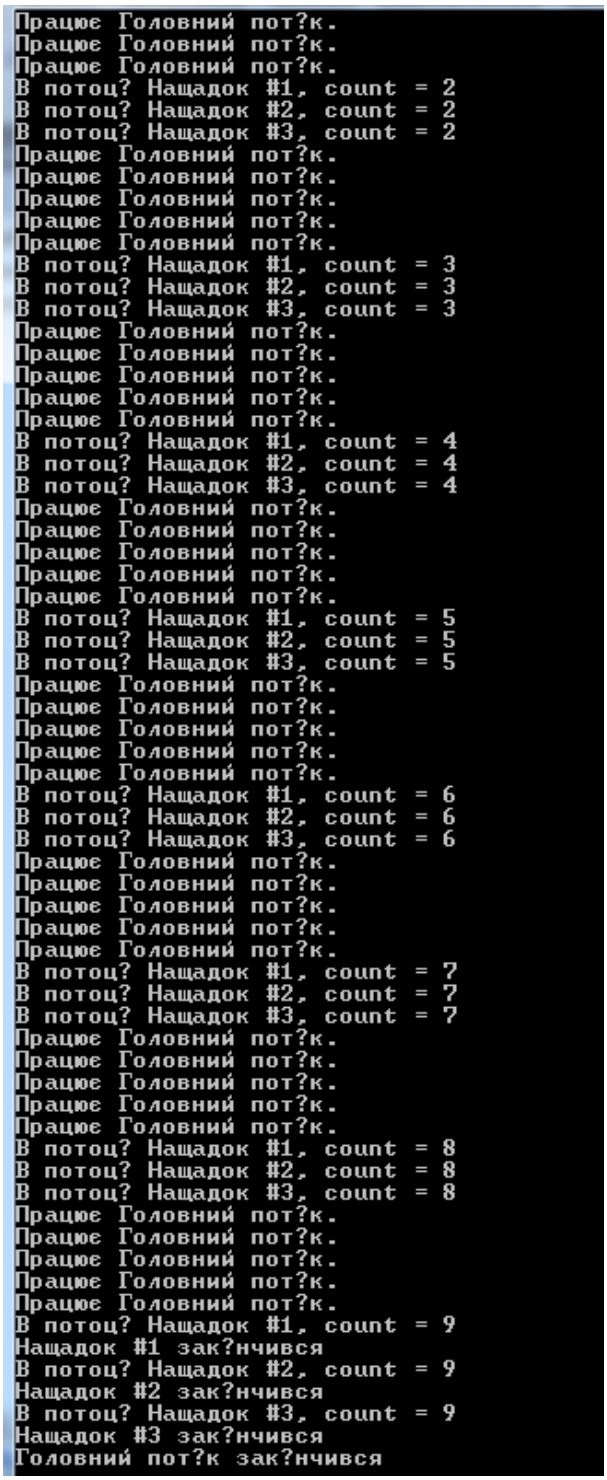
```
// Створити декілька потоків виконання.
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Reflection;
using System.Diagnostics;

namespace Lab1_Thread1
{
    class MyThread
    {
        public int count;
        public Thread Thrd;
        public MyThread(string name)
        {
            count = 0;
            Thrd = new Thread(this.Run);
            Thrd.Name = name; // задати ім'я потоку
            Thrd.Start(); // почати потік
        }
        // Точка входу в потік,
        public void Run() {
            Console.WriteLine(Thrd.Name + " почався");
            do {
                Thread.Sleep(500);
                Console.WriteLine("В потоці " + Thrd.Name + ", count = " + count);
                count++;
           } while(count < 10) ;
```

```
            Console.WriteLine(Thrd.Name + " закінчився");
        }
    }
    class MultiThread
    {
        static void Main() {
        Console.WriteLine("Головний потік почався");
        // Спочатку створит об'єкт класу MyThread.
        MyThread mt1 = new MyThread("Нащадок #1");
        MyThread mt2 = new MyThread("Нащадок #2");
        MyThread mt3 = new MyThread("Нащадок #3");
        do {
        Console.WriteLine("Працює Головний потік.");
       // призупинити головний потік на 500 мс для завершення виконання другого потоку
         Thread.Sleep(100);
        } while (mt1.count < 10 || mt2.count < 10 || mt3.count < 10);
        Console.WriteLine("Головний потік закінчився");
        Console.ReadKey();
      }
    }

}
```

Результат



Як бачите, після того, як всі три потоки почнуть виконуватися, вони спільно використовуватимуть ЦП. Наведений вище результат може відрізнятися залежно від середовища виконання, операційної системи і інших зовнішніх чинників, що впливають на виконання програми.

Висновки

Система багатопотокової обробки .NET Framework грунтується на класі Thread, який інкапсулює потік виконання. Клас Thread є герметичним, тобто він не може успадковуватися. У класі Thread визначений ряд методів і властивостей, призначених для управління потоками. В цій лекції ми розглянули деякі з них.

Метод **Start()** запускає потік на виконання.

Метод **Sleep ()** припиняє потік, *з якого він був викликаний* на певний період часу, що вказується в мілісекундах. Коли призупиняється один потік, може виконуватися іншій.

Контрольні питання і завдання для самостійної роботи

1. Яка різниця між процесом і потоком?
2. Що таке багатопотоковість?
3. Які ресурси потрібні для виконання програми?
4. Де знаходяться класи для роботи з потоками?
5. Який клас реалізує роботу з потоками?
6. Який потік є первинним? Як він створюється?
7. Як створити паралельний потік?
8. Скільки потоків можна створити в одному процесі? На які властивості роботи програми це вплине?
9. Як блокувати виконання потоку на певний час? Який метод для цього потрібно використати?
10. Коли виконання потоку завершується?
11. Який метод запускає вторинний потік?
12. Який метод запускає первинний потік?
13. Яке призначення статичного методу Sleep()?
14. Яким потоком повинна завершуватися робота програми: головним чи вторинним і чому?
15. Чи можна розмістити методи створення і запуску потоку в конструкторі класу, в якому описаний метод, що використовується в якості точки входу в потік?

Лабораторна робота 1.