Модуль 1. Багатопоточність

Лекція 3. Робота з потоками

Визначення моменту закінчення потоку. Передача аргумента потоку. Властивості потоку. Пріоритети потоків. Обчислення часу виконання потоків.

1. Визначення моменту закінчення потоку. Синхронізація потоків ча часом

Потік завершується коли виконаний останній оператор його методу і це не обов'язково головний потік (метод main).

Іноді важливо знати, коли саме завершується потік. У класі Thread є два способи для визначення моменту закінчення потоку: за допомогою властивості **IsAlive** і методу **Join()**..

Перший спосіб це використання булевої властивості **IsAlive**, яка визначена таким чином.

public bool IsAlive { get; }

Властивість IsAlive повертає логічне значення true, якщо потік, для якого вона викликається, як і раніше виконується. Для "випробування" властивості IsAlive в наступному прикладі потрібно в операторі циклу в методі main вказати цю властивість.

Приклад 1

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

namespace Lab2_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(100);
                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 100 мс для завершення потоків
                Thread.Sleep(100);
            } while (mt1.Thrd.IsAlive && mt2.Thrd.IsAlive && mt3.Thrd.IsAlive);

            Console.WriteLine("Головний потік закінчився");
            Console.ReadKey();
        }
    }

}
```

При виконанні цієї версії програми результат буде таким самим, як і раніше (у лекції 2). Єдина відмінність полягає в тому, що в ній використовується властивість **IsAlive** для відстежування моменту закінчення вторинних потоків.

Другий спосіб. Використання методу **Join ().**

Нижче наведена його проста форма.

public void Join()

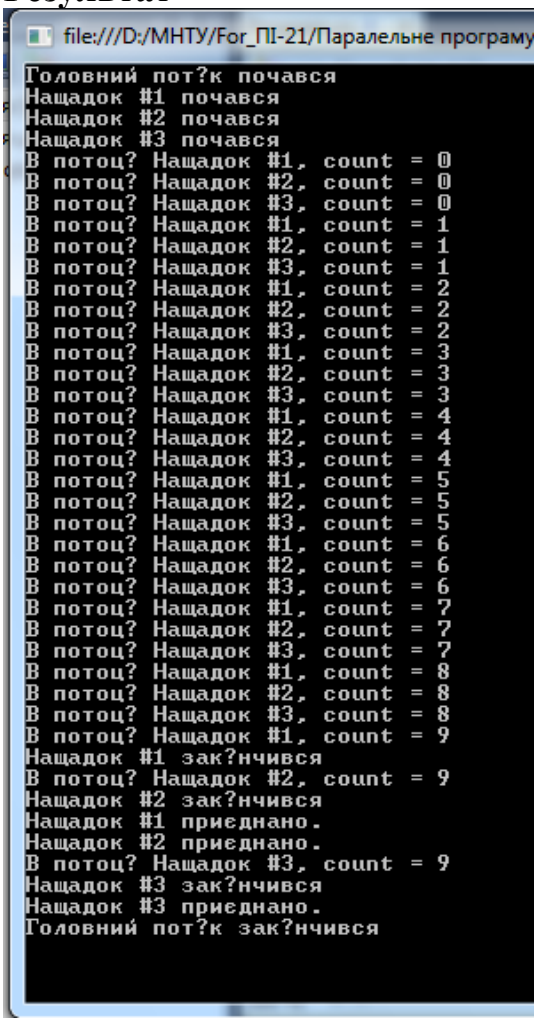
Метод Join () чекає до тих пір, поки потік, *для якого він був викликаний*, не завершиться. Його ім'я відображає принцип чекання до тих пір, поки потік, що викликає не приєднається до викликаного методу. Якщо ж даний потік не був початий, то генерується виключення ThreadStateException. У інших формах методу Join () можна вказати максимальний період часу, протягом якого слід чекати завершення вказаного потоку.

У наведеному нижче прикладі програми метод Join () використовується для того, щоб основний потік завершився останнім.

Приклад 2.

```
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(100);
                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");
             mt1.Thrd.Join();
             Console.WriteLine("Нащадок #1 приєднано.");
             mt2.Thrd.Join();
             Console.WriteLine("Нащадок #2 приєднано.");
             mt3.Thrd.Join();
             Console.WriteLine("Нащадок #3 приєднано.");
             Console.WriteLine("Головний потік закінчився");
             Console.ReadKey();
      }  } }
```

Результат



Як видно зі скриншоту, виконання потоків завершилося після повернення з послідовного ряду викликів методу Join ().

2. Передача аргумента потоку

Методи, які потребують розпаралелювання, можуть мати вхідні параметри (аргументи). Для передачі аргументу методу потоку потрібно використовувати перевизначену форму методу **Start ()**.

Аргумент передається потоку в наступній формі методу Start ().

public void Start(object параметр)

Об'єкт, що вказується як аргумент-параметр, автоматично передається методу, що виконує роль точки входу в потік. Отже, для того, щоб передати аргумент потоку, досить передати його методу Start ().

Для використання параметризованої форми методу Start () буде потрібна наступна форма конструктора класу Thread:

public Thread(ParameterizedThreadStart запуск)

де **запуск** позначає метод, що викликається з метою почати виконання потоку. Зверніть увагу на те, що в цій формі конструктора запуск має тип

ParameterizedThreadStart, а не **Threadstart**, як у формі, що використовувалася у попередніх прикладах. В даному випадку ParameterizedThreadStart є делегатом, що оголошується таким чином.

public delegate void ParameterizedThreadStart (object obj)

Як бачите, цей делегат приймає аргумент типа object. Тому для правильного використання такої форми конструктора класу Thread в методі, який використовується як точка входу в потік, має бути параметр типу **object**.

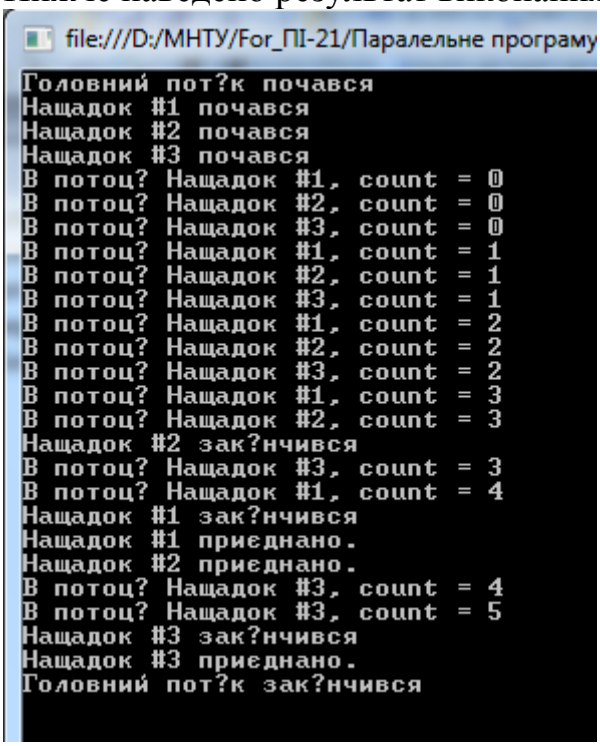
У наведеному нижче прикладі програми демонструється передача аргументу потоку.

Приклад 3.

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

namespace Lab2_Thread1
{
    class MyThread
    {
        public int count;
        public Thread Thrd;
        public MyThread(string name, int num)
        {
            count = 0;
            Thrd = new Thread(this.Run);
            Thrd.Name = name; // задати ім'я потоку
            Thrd.Start(num); // почати потік
        }
        // Точка входу в потік,
        public void Run(object num)
        {
            int num1 = (int)num;
            Console.WriteLine(Thrd.Name + " почався");
            do {
                Thread.Sleep(500);
                Console.WriteLine("В потоці " + Thrd.Name + ", count = " +
count);
                count++;
           } while(count < num1) ;
            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",5);
```

```
            MyThread mt2 = new MyThread("Нащадок #2",4);
            MyThread mt3 = new MyThread("Нащадок #3",6);
            mt1.Thrd.Join();
            Console.WriteLine("Нащадок #1 приєднано.");
            mt2.Thrd.Join();
            Console.WriteLine("Нащадок #2 приєднано.");
            mt3.Thrd.Join();
            Console.WriteLine("Нащадок #3 приєднано.");
            Console.WriteLine("Головний потік закінчився");
            Console.ReadKey();
    }  } }
```

Нижче наведено результат виконання програми.



Приклад 4.

В наступному прикладі створюється 3 потоки для виклику *трьох різних методів*. Для виклику перших двох методів потоки створюються в конструкторі класу. Для цього потрібно додати до класу конструктори. *Перший потік* викликає метод генерації простих чисел. Потік створюється і запускається в конструкторі класу. Другий потік викликає метод StudentHeight для створення масиву зросту студентів та його сортування. Він також створюється в конструкторі класу. Третій потік викликає метод генерації чисел Фібоначчі. Потік створюється і запускається в main().

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
namespace Lab_2
{
    class MyThread
    {
        public Thread thrd;
        public MyThread()
        {
            //пустий конструктор - тільки виділяє пам'ять
        }
        // Зверніть увагу на те, що конструктору класу
        // MyThread передається також значення типа int.
        public MyThread(string name, int num)
        {
            // Викликати конструктор типа ParameterizedThreadStart
            // явним чином лише для наочності прикладу.
            thrd = new Thread(SimpleNumber);
            thrd.Name = name;
            Console.WriteLine("Працює потік" + thrd.Name);
            // Тут змінна num передається методу Start ()
            //як аргумент.
            thrd.Start(num);
        }
        public MyThread(int num, string name)
        {
            // Викликати конструктор типа ParameterizedThreadStart
            // явним чином лише для наочності прикладу.
            thrd = new Thread(StudentHeight);
            thrd.Name = name;
            Console.WriteLine("Выполняется поток" + thrd.Name);
            // Тут змінна num передається методу Start ()
            //як аргумент.
            thrd.Start(num);
        }
        //генерация простых чисел
        public void SimpleNumber(object t)
        {
            int n = (int)t;
            bool[] table = new bool[n];
            int i, j;
            Thread.Sleep(500);  //приостанавливаем этот поток

            // Отмечаем все числа как простые
            for (i = 0; i < table.Length; i++)
                table[i] = true;
            // Вычеркиваем лишнее
            for (i = 2; i * i < table.Length; i++)
                if (table[i])
                    for (j = 2 * i; j < table.Length; j += i)
                        table[j] = false;
            // Выводим найденное
            for (i = 2; i < table.Length; i++)
```

```
        {
            if (table[i])
                Console.WriteLine(i);
        }
        Console.ReadKey();
    }
    public void StudentHeight(object l)
    {
        // рост студентов
        int len = (int)l;
        Thread.Sleep(500);  //приостанавливаем этот поток
        Console.WriteLine("Выполняется поток" + thrd.Name);
        Console.WriteLine("Рост студентов");
        int[] stHeight = new int[25];
        Random rd = new Random();
        for (int i = 0; i < stHeight.Length; i++)
            stHeight[i] = rd.Next(160, 190);
        Array.Sort(stHeight);
        for (int i = 0; i < stHeight.Length; i++)
            Console.WriteLine(stHeight[i]);
    }
    //-----------------------
    //Генерация чисел Фибоначчи
    public void GenFibonachi(object t)
    {
        int n = (int)t;
        int[] fibonachi = new int[n];
        fibonachi[0] = 1;
        fibonachi[1] = 1;
        for (int i = 2; i < n; i++)
        {
            fibonachi[i] = fibonachi[i - 2] + fibonachi[i - 1];
        }
        for (int i = 0; i < n; i++)
            Console.WriteLine("fibonachi[" + i + "]=" + fibonachi[i]);
    }
}
//--------------------

class Program
{
    static void Main(string[] args)
    {
        MyThread w0 = new MyThread(" #1", 10);  //перший конструктор
        MyThread w1 = new MyThread(20, " #2");  //другий конструктор
        MyThread w2 = new MyThread(); //третій конструктор без параметрів
        //потік для генерації чисел Фиб. створюється в main
        Thread thrd1;
        thrd1 = new Thread(w2.GenFibonachi);
        thrd1.Name = "Третій потік";
        Console.WriteLine("Працює потік" + thrd1.Name);
        // Тут змінна num передається методу Start () як аргумент.
        thrd1.Start(20);
        Thread.Sleep(500);  //призупиняємо цей потік
        Console.WriteLine("Главный поток завершен");
        Console.ReadKey();
```

```
        }
    }
}
```

3. Властивості потоків

3.1. Основні властивості потоків

В класі Thread визначено властивості потоку, деякі з них ми вже розгядали. Підсумуємо основні властивості потоку.

Таблиця 1. Властивості потоків

Властивість	**Опис**
Name	Ім'я потоку
ManagedThreadId	Номер потоку
IsAlive	Ознака існування потоку
ThreadState	Стан потоку
Priority	Пріоритет потоку
IsBackground	Ознака фонового потоку

Приклад 5. Фрагмент коду, який демонструє створення потоків та виведення інформації про них.

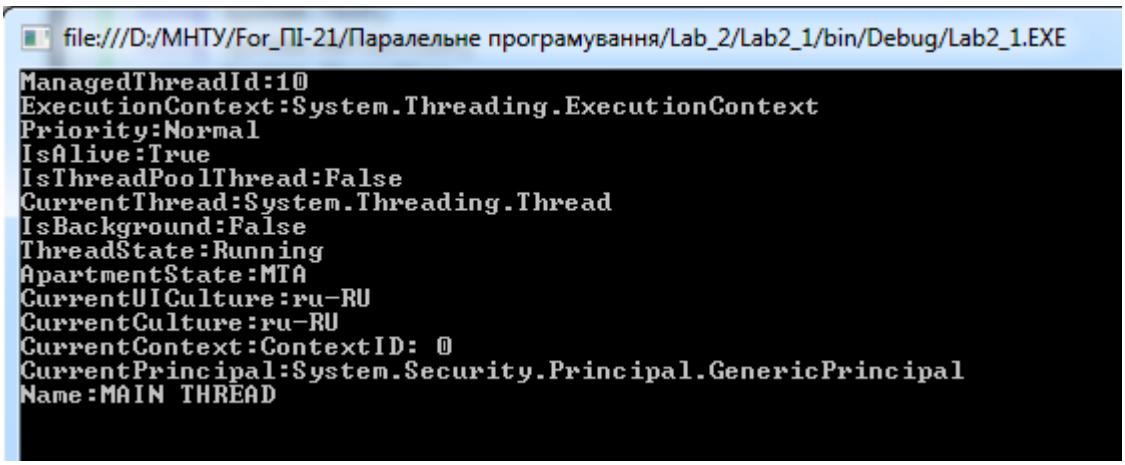
```
class Program
    {
        static void SomeFunc()
        {
          Console.WriteLine("Працює метод SomeFunc");
        }
        static void Main(string[] args)
        {
        // Оголошуємо масив потоків
       Thread[] arThr = new Thread[3];
       for(int i=0; i<arThr.Length; i++)
       {
         arThr[i] = new Thread(SomeFunc);
         arThr[i].Start();
       }
       for(int i=0; i<arThr.Length; i++)
       {
         // Виводимо інформацію про потоки
         Console.WriteLine("Thread Id: {0},name: {1}, IsAlive: {2}",
                arThr[i].ManagedThreadId,
                arThr[i].Name,
                arThr[i].IsAlive);
       }
       Console.ReadKey();
        }
    }
```

Властивості поточного потоку можна отримати за допомогою об'єкту Thread.CurrentThread. Приклад 6 демонструє виведення на консоль властивостей Головного потоку (main).

Приклад 6

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

namespace Lab2_1
{
    class Program
    {
        static void Main(string[] args)
        {
            Thread t = Thread.CurrentThread;
            t.Name = "MAIN THREAD";
            foreach (PropertyInfo p in t.GetType().GetProperties())
            {
                Console.WriteLine("{0}:{1}",
                    p.Name, p.GetValue(t, null));
            }
            Console.ReadKey();
        }
    }
}
```

Результат виконання:



3.2. Фоновий потік

У середовищі .NET Framework визначені два *типи потоків*: *пріоритетний* (або основний) і *фоновий*. Єдина відмінність між ними полягає в тому, що процес не завершиться до тих пір, поки не закінчиться *пріоритетний потік*, тоді як фонові потоки завершуються автоматично після закінчення всіх пріоритетних потоків. За замовчанням створюваний потік стає пріоритетним. Але його можна зробити фоновим, використовуючи властивість **IsBackground,** визначену в класі Thread, таким чином.

public bool IsBackground { get; set; }

Для того, щоб зробити потік фоновим, потрібно призначити логічне значення **true** властивості **IsBackground**. А логічне значення **false** вказує на те, що потік є пріоритетним.

Приклад 7. Робота основних і фонових потоків
(http://msdn.microsoft.com/ru-ru/library/system.threading.thread.isbackground.aspx)

Створюються основний (пріоритетний) і фоновий потоки. Основний потік утримує потік до завершення його циклу while. Після завершення основного потоку він закривається до того, як фоновий потік завершить свій цикл while.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
namespace Lab2
{
    class Test
    {
        //Робота основного і фонового потоків
        static void Main()
        {
            //створення об'єкту shortTest класу BackgroundTest
            //в конструктор класу передається кількість ітерацій - 10
            BackgroundTest shortTest = new BackgroundTest(10);
            //створення потоку foregroundThread для shortTest
            Thread foregroundThread =
                new Thread(new ThreadStart(shortTest.RunLoop));
            //призначення імені потоку
            foregroundThread.Name = "Пріоритетний потік";
            //створення об'єкту longTest класу BackgroundTest
            //в конструктор класу передається кількість ітерацій - 50
            BackgroundTest longTest = new BackgroundTest(50);
            //створення потоку backgroundThread для longTest
            Thread backgroundThread =
                new Thread(new ThreadStart(longTest.RunLoop));
            //призначення імені потоку
            backgroundThread.Name = "Фоновий потік";
            //встановлення для потоку властивості IsBackground
            backgroundThread.IsBackground = true;
            //запуск потоків
            foregroundThread.Start();
            backgroundThread.Start();
            Console.ReadKey();
        }
    }

    class BackgroundTest
    {
        int maxIterations;
        public BackgroundTest(int maxIterations)  //це конструктор класу
        {
            this.maxIterations = maxIterations;
        }
        public void RunLoop()
        {
            String threadName = Thread.CurrentThread.Name;  //ім'я потоку
```

```
            for (int i = 0; i < maxIterations; i++)
            {
                Console.WriteLine("{0} count: {1}", threadName,
i.ToString());
                Thread.Sleep(250);
            }
            Console.WriteLine("{0} завершення потоку:  ", threadName);
        }
    }

}
```

3.3. Призначення пріоритетів потокам

Кожний потік має свій пріоритет, який визначає, наскільки часто потік дістає доступ до процесора. Потоки з низьким пріоритетом дістають доступ до процесора рідше, ніж з високим. Таким чином, протягом заданого проміжку часу потоку з низьким пріоритетом буде виділено менше процесорного часу, ніж потоку з високим пріоритетом. Процесорний час, що отримується потоком, робить визначальний вплив на виконання потоку і його взаємодії з іншими потоками, які виконуються паралельно. Слід мати на увазі, що, крім пріоритету, на частоту доступу потоку до процесора впливають і інші чинники. Так, якщо потік з високим пріоритетом чекає доступу до деякого ресурсу, наприклад для введення з клавіатури, він блокується, а замість нього виконується низькопріоритетний потік. У подібній ситуації потік з низьким пріоритетом може діставати доступ до процесора частіше, ніж високопріоритетний потік протягом певного періоду часу. І нарешті, конкретне планування завдань на рівні операційної системи також впливає на час процесора, що виділяється для потоку.

Коли створений потік починає виконуватися, він отримує пріоритет, що встановлюється за замовчанням. Пріоритет потоку можна змінити за допомогою властивості **Priority**, яка є членом класу **Thread**. Нижче наведена загальна форма цієї властивості:

public ThreadPriority Priority{ get; set; }

де ThreadPriority позначає перелічування, в якому визначаються наведені нижче значення пріоритетів.

ThreadPriority.Highest
ThreadPriority.AboveNormal
ThreadPriority.Normal
ThreadPriority.BelowNormal
ThreadPriority.Lowest

За замовчанням для потоку встановлюється значення пріоритету **ThreadPriority.Normal.**

Вплив пріоритетів позначається тільки у разі конкуренції потоків за потужності ЦП (наприклад, якщо кількість ядер менша ніж кількість потоків).

Для того, щоб став зрозумілішим вплив пріоритетів на виконання потоків, розглянемо приклад, в якому виконуються два потоки: один з яких з вищим пріоритетом. Обидва потоки створюються як екземпляри об'єктів класу MyThread.

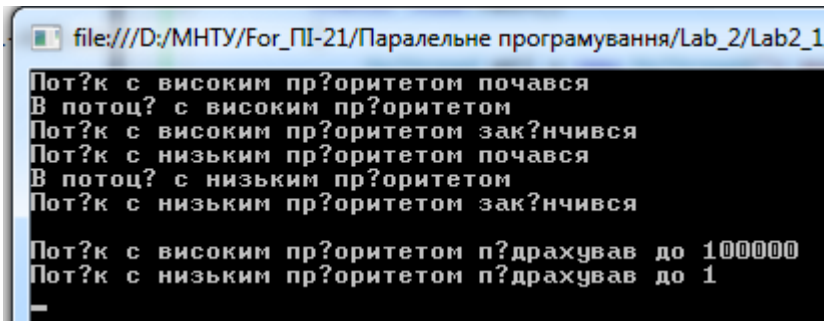
У методі Run () організується цикл, в якому підраховується певне число повторень. Цикл завершується, коли підрахунок досягає величини 100000 або коли статична змінна **stop** набуває логічного значення **true**. Спочатку змінна stop набуває логічного значення false. У першому потоці, де робиться підрахунок до 100000, змінній stop призначається значення **true** . Через це другий потік закінчується на наступному своєму інтервалі часу. На кожному кроці циклу рядок у змінній currentName перевіряється на наявність імені виконуваного потоку. Якщо імена потоків не збігаються, це означає, що сталося перемикання виконуваних завдань. Всякий раз, коли відбувається перемикання завдань, ім'я нового потоку відображається і призначається змінній currentName. Це дає можливість відстежити частоту доступу потоку до процесора. Після закінчення обох потоків на консоль виводиться число повторень циклу в кожному з них.

Приклад 8

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

namespace Lab2
{
    class MyThread
    {
        public int count;
        public Thread Thrd;
        static bool stop = false;
        static string currentName;
        /* Сконструювати новий потік. Зверніть увагу на те, що
        конструктор ще не починає виконання потоків. */

        public MyThread(string name)
        {
            //це конструктор класу з параметром name
            count = 0;
            Thrd = new Thread(this.Run);
            Thrd.Name = name;
            currentName = name;
        }
        // Почати виконання нового потоку
        void Run()
        {
            Console.WriteLine("Потік " + Thrd.Name + " почався");
            do
            {
                count++;
                if (currentName != Thrd.Name)
```

```
                {
                    currentName = Thrd.Name;
                    Console.WriteLine("В потоці " + currentName);
                }
            } while (stop == false && count < 100000);
            stop = true;
            Console.WriteLine("Потік " + Thrd.Name + " закінчився");
        }
    }
    class PriorityDemo
    {

        static void Main()
        {
            MyThread mt1 = new MyThread("с високим пріоритетом");
            MyThread mt2 = new MyThread("с низьким пріоритетом");
            // Встановити пріоритети для потоків.
            mt1.Thrd.Priority = ThreadPriority.AboveNormal;
            mt2.Thrd.Priority = ThreadPriority.BelowNormal;
            // Почати потоки,
            mt1.Thrd.Start();
            mt2.Thrd.Start();
            mt1.Thrd.Join();
            mt2.Thrd.Join();
            Console.WriteLine();
            Console.WriteLine("Потік " + mt1.Thrd.Name + " підрахував до " +
mt1.count);
            Console.WriteLine("Потік " + mt2.Thrd.Name + " підрахував до " +
mt2.count);
            Console.ReadKey();
        }
    }

}
```

Результат виконання:



Якщо подивитися на результати, потік з високим пріоритетом отримав більше часу на виконання, ніж потік з низьким пріоритетом.

Конкретний результат може відрізнятися залежно від швидкодії процесора і кількості інших завдань, що вирішуються в системі, а також від використовуваної версії Windows.

Багатопотоковий код може поводитися по-різному в різних середовищах, тому ніколи не слід покладатися на результати його виконання лише в одному

середовищі. Так, було б помилкою вважати, що низькопріоритетний потік з наведеного вище прикладу завжди виконуватиметься лише протягом невеликого періоду часу до тих пір, поки не завершиться високопріоритетний потік. У іншому середовищі високопріоритетний потік може, наприклад, завершитися ще до того, як низькопріоритетний потік виконається хоча б один раз.

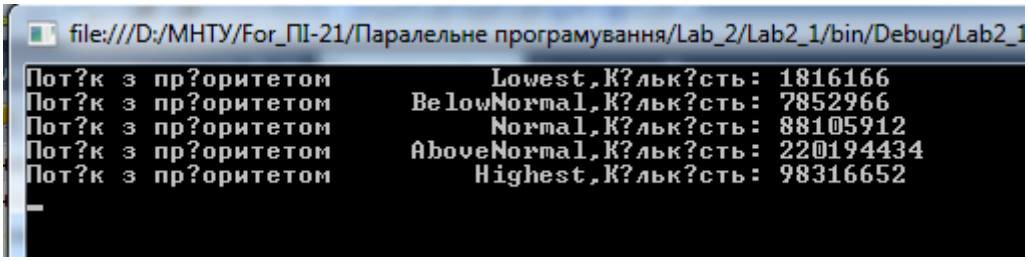
Приклад 9

http://www.intuit.ru/studies/courses/5938/1074/lecture/16447?page=3

У наступному фрагменті п'ять потоків з різними пріоритетами конкурують за доступ до процесора з двома ядрами. Кожен потік збільшує свій лічильник.

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

namespace Lab2
{
    class Program
    {
        static long[] counts;
        static bool finish;
        static void ThreadFunc(object iThread)
        {
            while (true)
            {
                if (finish)
                    break;
                counts[(int)iThread]++;
            }
        }

        static void Main(string[] args)
        {
            counts = new long[5];
            Thread[] t = new Thread[5];
            for (int i = 0; i < t.Length; i++)
            {
                t[i] = new Thread(ThreadFunc);
                t[i].Priority = (ThreadPriority)i;
            }
            // Запускаємо потоки
            for (int i = 0; i < t.Length; i++)
                t[i].Start(i);

            // Даємо потокам попрацювати поработать 1 с
            Thread.Sleep(1000);
            // Сигнал про завершення
            finish = true;

            // Чекаємо завершення усіх потоків
            for (int i = 0; i < t.Length; i++)
                t[i].Join();
            // Виводимо результати
            for (int i = 0; i < t.Length; i++)
                Console.WriteLine("Потік з пріоритетом {0, 15},Кількість: {1}",
(ThreadPriority)i, counts[i]);
```

```
            Console.ReadKey();
        }
    }

}
```



4. Обчислення часу виконання програми

Для визначення ефективності розпаралелювання програми потрібно вміти визначати час роботи. Це можна зробити за допомогою класу DateTime або класу Stopwatch.

4.1. Обчислення часу виконується за допомогою методів класу DateTime.

Таблиця 2. Клас DateTime і його члени

Властивість	**Призначення**
Hour	Повертає компонент години дати, представленої цим екземпляром.
Minute	Повертає компонент хвилини дати, представленої цим екземпляром.
Second	Повертає компонент секунди дати, представленої цим екземпляром
Millisecond	Повертає компонент мілісекунд для дати, представленої в даному екземплярі.
Now	Повертає об'єкт DateTime, якому привласнені поточна дата і час даного комп'ютера, виражені як місцевий час.
Ticks	Повертає число тактів, яке представляє дату і час цього екземпляра.

Приклад. Обчислення часу роботи програми з використанням DateTime

```
DateTime dt1, dt2;
dt1 = DateTime.Now;
// Виклик_обчислювальної_процедури;

dt2 = DateTime.Now;
TimeSpan ts = dt2 - dt1;
Console.WriteLine("Total time: {0}", ts.TotalMilliseconds);
```

4.2. Обчислення часу роботи програми з використанням класу Stopwatch

Також можна використовувати об'єкт Stopwatch простору System.Diagnostics:

Приклад. Обчислення часу роботи програми з використанням класу Stopwatch

```
Stopwatch sw = new Stopwatch();
sw.Start();
// Виклик_обчислювальної_процедури;
sw.Stop();
TimeSpan ts = sw.Elapsed;
Console.WriteLine("Total time: {0}", ts.TotalMilliseconds);
```

При оцінці продуктивності необхідно врахувати, що час виконання алгоритму залежить від багатьох параметрів. Тому бажано оцінювати середній час виконання при декількох прогонах алгоритму, виключаючи перший прогін.

Висновки

Система багатопотчної обробки 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.

Метод **Start()** запускає потік на виконання. Для передачі параметру в потік використовується параметризована форма методу **Start(param).**

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

Властивість **IsAlive** призначена для відстежування моменту закінчення породжених потоків (можливі значення true, false).

Метод **Join ()** призупиняє потік до тих пір, поки вказаний потік, *для якого він був* викликаний, не завершиться.

У середовищі .NET Framework визначені два *типи потоків*: пріоритетний (або основний) і фоновий. За замовчанням створюваний потік стає пріоритетним, але його можна зробити фоновим, використовуючи властивість **IsBackground** (для цього досить призначити їй логічне значення **true).** А логічне значення **false** вказує на те, що потік є пріоритетним.

Пріоритет потоку можна змінити за допомогою властивості **Priority**, яка є членом класу **Thread**.

Контрольні запитання і завдання для самостійного виконання.

1. В якому просторі імен визначені класи, що підтримують багатопотокове програмування?
2. У якому класі визначені методи і властивості для управління потоками?
3. Який метод запускає вторинний потік?
4. Який метод запускає первинний потік?
5. Яке призначення статичного методу Sleep()?

6. Яким потоком повинна завершуватися робота програми: головним чи вторинним і чому?
7. Чи можна розмістити методи створення і запуску потоку в конструкторі класу, в якому описаний метод, що використовується в якості точки входу в потік?
8.Як передати в метод паралельного потоку параметри?
9. Скільки паралельних потоків можна створити в процесі програми?
10. Яке призначення властивості IsAlive і що означають її значення?
11. Яке призначення методу Join ()? Коли його варто використовувати?
12. Які основні властивості потоків?
13. За допомогою якого об'єкту можна отримати властивості поточного потоку?
14. Як встановити пріоритети потокам?
15. Оцінити час роботи послідовної і паралельної версії програми генерації масиву чисел та його сортування.
16. Те саме зробити для свого варіанту завдання.
Лабораторна робота 2