**Кривонос М.О., Кривонос О.М.**
*Житомирський державний університет імені Івана Франка*
*м. Житомир, Україна*
*misha@zu.edu.ua, krypton@zu.edu.ua*

# ПРИКЛАД РЕАЛІЗАЦІЇ ЗАДАЧІ З ВИКОРИСТАННЯМ БАГАТОПОТОЧНОСТІ НА МОВІ C#

*The article provides an example of C# coded tools for implementing the use of multithreading in solving the problem of pinging a computer network.*

***Keywords:*** *multithreading, computer network, C#.*

Мова C# може використовується для написання програм, призначених для кінцевого користувача, при цьому досить часто виникає необхідність звертатися до пристроїв введення-виведення, баз даних, або до мережевих ресурсів [1] Ефективно вирішувати подібні завдання дозволяє багатопоточність.

Багатопоточність – властивість коду програми виконуватися паралельно (одночасно) на кількох ядрах процесора або виконуватися псевдопараллельно на одному ядрі (кожен потік отримує в своє розпорядження деякий час, за яке він встигає виконати певну частину свого коду на процесорі).

Потік (thread) являє собою незалежну послідовність інструкцій в програмі. У додатках, які мають користувальницький інтерфейс, завжди є як мінімум один головний потік, який відповідає за стан компонентів інтерфейсу. Крім нього в програмі може створюватися безліч незалежних дочірніх потоків, які будуть виконуватися самостійно [2].

Зрозуміло, немає сенсу розділяти завдання, швидкість виконання яких залежить тільки від швидкості процесора, однак якщо в програмі

присутні запити введення, запити з мережевих джерел і будь-яких сторонніх програм, процесор змушений простоювати і цей час витрачається нераціонально. У таких випадках необхідно розділити виконання програми на кілька потоків

Як приклад розглянемо просту «пінговалку» - програму, яка визначає діючі комп'ютери в підмережі. Робимо це наступним чином: комп'ютери, що входять в одну підмережу, належать до одного діапазону IP адрес, адреса, що закінчується нулем, є адресою підмережі, а як що закінчується числом 255, то ді вона є широкомовною адресою даної мережі. Адреси в діапазоні від 1 до 254 включно є повноправними адресами хостів всередині підмережі, їх можна використовувати для призначення на комп'ютерах, тому зчитуємо адресу мережі, замінюємо останню цифру 0 на число від 1 до 254, щоб отримати адресу, яку може мати комп'ютер обраної підмережі (рис. 1.)

```
static void Main(string[] args)
{
    //Створюємо об’єкт  Ping
    System.Net.NetworkInformation.Ping ping =
    new System.Net.NetworkInformation.Ping();
    //Зчитуємо адресу підмережм
    string str = Console.ReadLine();
    str = str.Substring(0, str.Length - 1);
    //Перевіряємо всі адреси мережи
    for (int i = 1; i < 255; ++i)
    {
        System.Net.NetworkInformation.PingReply pingReply =
        ping.Send(str + i, 1000);
        if (pingReply.Status.ToString().Equals("Success"))
            Console.WriteLine(str + i);
    }
}
```

Рис. 1 – Скролінг коду, що реалізує процес сканування мережі.

Ця програма працює і дійсно виводить адреси всіх підключених комп'ютерів, адже вона перевіряє весь діапазон IP адрес підмережі на доступність. Однак, таких адрес 254 і для кожного програма чекає відгуку протягом секунди. Очевидно, що незалежно від потужності процесора час

виконання такої програми може становити до 254 секунд. Для користувальницького додатка таке неприпустимо. Тут нам і приходить на допомогу багатопоточність.

Ми можемо пінгувати кожен комп'ютер паралельно, розділивши процес виконання на 254 потоків. Для цього ми будемо використовувати вбудовану в C # бібліотеку System.Threading. Щоб запустити новий потік нам спочатку необхідно його створити і передати йому на виконання функцію, яка повертає void (рис. 2). В найпростішому вигляді створення потоку виглядає так:

```
static void Main(string[] args)
{
    //Створюємо об'єкт потоку
    Thread thread = new Thread(ThreadFunction);
    //Запускаємо потік
    thread.Start();
}
//Функція потоку
static void ThreadFunction()
{
    //Код функції
}
```

Рис. 2. – Створення потоку

Клас Thread визначає ряд методів і властивостей, які дозволяють керувати потоком і отримувати дані про нього.

Деякі методи класу Thread:

1. Статичний метод Sleep зупиняє потік на певну кількість мілісекунд.

2. Метод Abort повідомляє середу CLR про те, що треба припинити потік.

3. Метод Interrupt перериває потік на деякий час.

4. Метод Join блокує виконання викликав його потоку до тих пір, поки не завершиться потік, для якого був викликаний даний метод.

5. Метод Resume відновлює роботу раніше припиненого потоку.

6. Метод Start запускає потік.

7. Метод Suspend припиняє потік [3].

Метод Start () має можливість передавати об'єкт з даними, що використовуються функцією в потоці. Зверніть увагу, що ми можемо створити функцію тільки з одним аргументом і тільки типу object, навіть якщо він є об'єктом int, як в нашому випадку. Таким чином ми можемо створити функцію з параметром і автоматично в циклі паралельно запускати функції з різними даними (рис. 3).

```
static void Main(string[] args)
{
    str = Console.ReadLine();
    str = str.Substring(0, str.Length - 1);
    for (int i = 1; i < 255; ++i)
    {
        //Створюємо потік
        Thread thread = new Thread(Function);
        //Запускаємо потік thread.Start(i);
    }
}
//Створюємо окрему функцію для запиту
static void Function(object i)
{
    System.Net.NetworkInformation.Ping ping =
     new System.Net.NetworkInformation.Ping();
    System.Net.NetworkInformation.PingReply pingReply =
    ping.Send(str + i, 1000);
    if (pingReply.Status.ToString().Equals("Success"))
        Console.WriteLine(str + i);
}
```

Рис. 3 – Скролінг коду, що реалізує процес сканування мережі, з застосуванням багато поточності.

Така програма виконується майже миттєво: процесор дуже швидко створив низку потоків, кожен з яких почав виконуватися відразу ж після створення, відправив запит за адресою і отримав відповідь, не затримуючи інші потоки. Тобто за допомогою використання багатопоточності ми прискорили виконання програми в 254 разів. А якщо уявити, що необхідно працювати з величезною базою даних, і безліч разів запитувати дані, поки

користувач працює з програмою, стає очевидно, що без поділу роботи не обійтися. Багатопоточність незамінна тоді, коли необхідно, щоб графічний інтерфейс продовжував відгукуватися на дії користувача під час виконання деякої обробки інформації, або коли програма виконує безліч дій, для яких потрібні дані зі сторонніх джерел. Таким чином, використання багатопоточності в програмуванні є не просто методом оптимізації, а необхідністю.

ЛІТЕРАТУРА

1. Александров Э. Э., Афонин В. В. Программирование на языке C в Microsoft Visual Studio 2010. [Электронный ресурс]. Режим доступа: http://www.intuit.ru/department/ pl/prcmsvs2010/ (дата обращения: 01.11.2015).

2. Александров Э. Э., Афонин В. В. Введение в программирование в языке C. Саранск: Мордовский гос. университет им. Н. П. Огарева, 2009.

3. Джеффри Рихтер. CLR via C#. Программирование на платформе Microsoft. NET Framework 4.5 на языке C#. 4-е изд. СПб.: Питер, 2013.

**Кубик В.О., Толюпа С.В.**

*Київський національний університет імені Тараса Шевченка*

*м. Київ, Україна*

*cypherfounder@gmail.com, tolupa@i.ua*

## БАГАТОФАКТОРНА АВТЕНТИФІКАЦІЯ З ЗАСТОСУВАННЯМ БІОМЕТРІЇ ЯК МЕТОД ЗАХИСТУ ОБ'ЄКТІВ КРИТИЧНОЇ ІНФРАСТРУКТУРИ

*Biometric authentication - authentication based on something that is an integral part of the authorized entity (its biometric parameter). The identifier in*