МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Ректор НУХТ

_____ Олександр ШЕВЧЕНКО

«___» _____ 2023 р.

ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ

Лабораторний практикум

для здобувачів освітнього ступеня «Бакалавр»
спеціальності 122 «Комп'ютерні науки»
освітньо-професійних програм «Комп'ютерні науки»
та «Інформаційні системи та штучний інтелект»
денної та заочної форм навчання

Всі цитати, цифровий та фактичний матеріал, бібліографічні відомості перевірені. Написання одиниць відповідає стандартам

Підписи авторів:

_____ Микола КОСТІКОВ

СХВАЛЕНО
на засіданні кафедри
інформаційних технологій,
штучного інтелекту і
кібербезпеки
Протокол № 11
Від 07.02.2023 р.

Реєстраційний номер
електронних методичних
рекомендацій у НМУ

50.103-2023

КИЇВ НУХТ 2023

Паралельне програмування [Електрон. ресурс]: лабораторний практикум для здобувачів освітнього ступеня «Бакалавр» спеціальності 122 «Комп'ютерні науки» освітньо-професійних програм «Комп'ютерні науки» та «Інформаційні системи та штучний інтелект» денної та заочної форм навчання / уклад. : М. П. Костіков. – К. : НУХТ, 2023. – 35 с.

Рецензент: **О. В. Харкянен,** канд. техн. наук, доц.

Укладач: **М. П. Костіков,** канд. техн. наук, доц.

Відповідальний за випуск: **С. В. Грибков**, д-р техн. наук, проф.

Подано в авторській редакції

ЗМІСТ

ВСТУП

Метою виконання лабораторних робіт є закріплення у здобувачів знань і вмінь із дисципліни «Паралельне програмування», набуття навичок використання сучасних технологій паралельного програмування для подальшого їх використання у своїй професійній діяльності.

Завданням лабораторного практикуму є допомога здобувачам в опануванні методів паралельного програмування, які розглядаються в рамках навчальної дисципліни. В цьому практикумі викладено загальні рекомендації щодо створення паралельних програм, наведено короткі теоретичні відомості, завдання та індивідуальні варіанти для виконання лабораторних робіт, запитання для самоконтролю та рекомендовану літературу.

Лабораторні роботи з дисципліни «Паралельне програмування» розроблено відповідно до робочої програми навчальної дисципліни.

Методичні вказівки складаються з шести лабораторних робіт, у яких послідовно описано різні типи завдань із паралельного програмування, розглянутих у рамках навчальної дисципліни.

Виконання кожної лабораторної роботи передбачає ознайомлення здобувачів із методичними вказівками та теоретичну підготовку з відповідних розділів дисципліни «Паралельне програмування».

Для виконання лабораторних робіт при створенні паралельних програм можна використовувати довільні мови програмування, які підтримують багатопоточність (зокрема C#, Java, Python, C++ версії 11 і вище і т. д.), а також довільні середовища розроблення програмних засобів (IDE).

Для здачі лабораторної роботи здобувач має оформити звіт, який містить:

- титульний аркуш;
- формулювання завдання;
- індивідуальний варіант;
- опис ходу виконання роботи;
- висновки.

Бали, отримані за виконання лабораторних робіт, підсумовуються та враховуються при виставленні підсумкової оцінки з навчальної дисципліни.

ЛАБОРАТОРНА РОБОТА 1.

Створення та використання потоків для паралельних обчислень

Мета: навчитися створювати багатопоточні програми, керувати роботою потоків та реалізовувати паралельні обчислення.

Завдання

Написати консольну програму довільною мовою програмування (C#, Python, Java, C++, інші) з використанням потоків для паралельних обчислень.

Числа ***A, B, C, D, E, F*** підставити у завдання відповідно до свого варіанту (він визначається як порядковий номер студента в загальному списку групи). Значення по варіантах наведено в табл. 1.

1. При запуску програми вивести вітання та інформацію про автора (ПІБ, група).
2. Створити у програмі A потоків (не рахуючи головного). Назва кожного потоку повинна містити його номер.
3. Встановити режим виконання всіх потоків як «основний» («звичайний», «переднього плану», тобто не фоновий).
4. Створити для кожного потоку окремі методи (процедури) для реалізації дій у потоці.
5. У кожному методі на початку виводити в консоль повідомлення «Потік №... почав роботу», а в кінці — «Потік №... завершив роботу» (або інші аналогічні коментарі).
6. У потоці № A зробити затримку між початком і кінцем виконання на B секунд.
7. Реалізувати паралельне обчислення формули C:
   - розбити ряд на 2 рівні частини по D елементів кожна (через один / ін.чином);
   - одну з цих частин має рахувати потік № E, а іншу — потік № F.
8. Запустити на паралельне виконання всі створені потоки від № 1 до № A.
9. Після кінця обчислень у потоках № E і № F вивести загальну суму ряду в головному потоці з коментарем: «Результат обчислення формули ...: ...»
10. Перевірити отриманий результат на правильність (точність) за допомогою вбудованих математичних функцій чи формул обчислення суми арифметичної прогресії тощо (вивести результат їх роботи і порівняти).
11. Після цього в головному потоці перевірити стан потоку № A та вивести відповідний коментар на кшталт: «Потік № A ще працює» або «Потік № A вже не працює».
12. Якщо потік № A ще працює, примусово завершити його та вивести повідомлення в головному потоці, що потік завершено достроково.
13. Вивести в головному потоці повідомлення «Кінець головного потоку».

Табл. 1. Індивідуальні варіанти до лабораторної роботи № 1

№	A	B	C	D	E	F
1	4	11	$4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 ... = \pi$	125000	1	2
2	4	12	$1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 ... = \pi^2 / 6$	18000	1	3
3	4	13	$1/1^2 - 1/2^2 + 1/3^2 - 1/4^2 ... = \pi^2 / 12$	15000	2	3
4	5	14	$1/1^2 + 1/3^2 + 1/5^2 + 1/7^2 ... = \pi^2 / 8$	12000	1	2
5	5	15	1/(1*2) + 1/(2*3) + 1/(3*4) + 1/(4*5)... = 1	9000	1	3
6	5	16	1 + 2 + 3 + 4 + 5 + 6 ...	200000	1	4
7	5	17	1 - 2 + 3 - 4 + 5 - 6 ...	180000	2	3
8	5	18	1 + 3 + 5 + 7 + 9 + 11 ...	160000	2	4
9	5	19	1 - 3 + 5 - 7 + 9 - 11 ...	140000	3	4
10	6	20	1 + 4 + 7 + 10 + 13 + 16 ...	120000	1	2
11	6	11	1 - 4 + 7 - 10 + 13 - 16 ...	100000	1	3
12	6	12	$1/1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 ... = \pi / 4$	250000	1	4
13	6	13	$4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 ... = \pi$	150000	1	5
14	6	14	$1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 ... = \pi^2 / 6$	16000	2	3
15	6	15	$1/1^2 - 1/2^2 + 1/3^2 - 1/4^2 ... = \pi^2 / 12$	13000	2	4
16	6	16	$1/1^2 + 1/3^2 + 1/5^2 + 1/7^2 ... = \pi^2 / 8$	10000	2	5
17	6	17	1/(1*2) + 1/(2*3) + 1/(3*4) + 1/(4*5)... = 1	7000	3	4
18	6	18	1 + 2 + 3 + 4 + 5 + 6 ...	80000	3	5
19	6	19	1 - 2 + 3 - 4 + 5 - 6 ...	70000	4	5
20	7	20	1 + 3 + 5 + 7 + 9 + 11 ...	60000	1	2
21	7	11	1 - 3 + 5 - 7 + 9 - 11 ...	50000	1	3
22	7	12	1 + 4 + 7 + 10 + 13 + 16 ...	40000	1	4
23	7	13	1 - 4 + 7 - 10 + 13 - 16 ...	30000	1	5
24	7	14	$1/1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 ... = \pi / 4$	220000	1	6
25	7	15	$4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 ... = \pi$	175000	2	3
26	7	16	$1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 ... = \pi^2 / 6$	14000	2	4
27	7	17	$1/1^2 - 1/2^2 + 1/3^2 - 1/4^2 ... = \pi^2 / 12$	11000	2	5
28	7	18	$1/1^2 + 1/3^2 + 1/5^2 + 1/7^2 ... = \pi^2 / 8$	18000	2	6
29	7	19	1/(1*2) + 1/(2*3) + 1/(3*4) + 1/(4*5)... = 1	5000	3	4
30	7	20	1 + 2 + 3 + 4 + 5 + 6 ...	90000	3	5

Теоретичні відомості

Паралельні обчислення застосовуються для наступних цілей:

- одночасне виконання кількох задач на різних ядрах процесора, розподіл задач між ядрами:
  - математичні обчислення (ряди etc.)
  - опрацювання зображень (окремі пікселі)
  - інше
- паралельне виконання кількох задач у межах однієї програми;
- паралельне виконання багатьох процесів у рамках ОС.

Потік (потік виконання; нитка; англ. *thread*) — найменша частина програми, якою може керувати планувальник ОС.

Зазвичай потік є частиною процесу.

Потоки мають спільний адресний простір (у рамках процесу).

Потоки підтримуються в більшості сучасних мов програмування, наприклад:

- Java;
- C#;
- C++ (починаючи з C++11);
- Python;
- Ruby;
- Delphi;
- та інші.

Приклад коду на C# для створення першої найпростішої програми з одним потоком (головним, *Main*):

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello, world!");
        Console.Read();
    }
}
```

Приклад коду на C# для створення та запуску нового потоку (порожнього):

```
using System;
using System.Threading;

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello, world!");
        Thread t1 = new Thread(Actions1);
        t1.Start();
        Console.Read();
    }

    static void Actions1()
    {
    }
}
```

Приклад коду на C# для створення та запуску потоку з діями всередині:

```
using System;
using System.Threading;

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello, world!");
        Thread t1 = new Thread(Actions1);
        t1.Start();
        Console.Read();
    }

    static void Actions1()
    {
        Console.WriteLine("t1 starts");
        Console.WriteLine("(Actions inside t1)");
        Console.WriteLine("t1 ends\n");
    }
}
```

Приклад коду на C# для створення і запуску потоку № 2 паралельно з № 1:

```
...
    static void Main()
    {
        Console.WriteLine("Hello, world!");
        Thread t1 = new Thread(Actions1);
        Thread t2 = new Thread(Actions2);
        t1.Start();
        t2.Start();
        Console.Read();
    }
...
    static void Actions2()
    {
        Console.WriteLine("t2 starts");
        Console.WriteLine("(Actions inside t2)");
        Console.WriteLine("t2 ends\n");
    }
}
```

Приклад коду на C# із доданням навантаження на перший потік:

```
...
static void Actions1()
    {
        Console.WriteLine("t1 starts");
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine("(Actions inside t1)");
        }
        Console.WriteLine("t1 ends\n");
    }

    static void Actions2()
    {
        Console.WriteLine("t2 starts");
        Console.WriteLine("(Actions inside t2)");
        Console.WriteLine("t2 ends\n");
    }
}
```

Приклад коду на C# для призупинення (сну) потоку:

```
static void Actions1()
    {
        Console.WriteLine("t1 starts");
        Thread.Sleep(1000);
        Console.WriteLine("(Actions inside t1)");
        Thread.Sleep(1000);
        Console.WriteLine("t1 ends\n");
    }
```

Приклад коду на C# для перевірки стану потоку:

```
static void Main()
  {
    Console.WriteLine("Hello, world!");
    Thread t1 = new Thread(Actions1);
    t1.Start();
    while (t1.IsAlive)
    {
      Console.WriteLine("t1 is alive");
      Thread.Sleep(200);
    }
    Console.Read();
  }
```

Висновки

У ході виконання лабораторної роботи визначено, яким чином можна створювати паралельні потоки виконання програми та керувати ними. З’ясовано, що при одночасному запуску потоків послідовність їхнього виконання є непередбачуваною.

Контрольні запитання

1. Що таке потік?
2. Для чого створюються багатопоточні програми?
3 Яким чином процесор працює з багатьма потоками? (одночасність та інше)
4. Різниця у роботі одно- та багатоядерного процесора.
5. Що таке метод Join()? Як він працює?
6. Які бувають режими виконання потоків? У чому різниця між ними?

Література

Див. джерела №№ 1, 3, 4, 5.

ЛАБОРАТОРНА РОБОТА 2.

Вимірювання прискорення та ефективності паралельних обчислень

Мета: навчитися вимірювати час, потрібний для виконання потоків паралельної програми, а також обчислювати прискорення та ефективність багатопоточних програм, у яких реалізовано паралельні обчислення.

Завдання

Написати консольну програму довільною мовою програмування (C#, Python, Java, C++, інші) з використанням потоків для паралельних обчислень.

Формулу ***A*** підставити у завдання відповідно до свого варіанту (він визначається як порядковий номер студента в загальному списку групи). Значення по варіантах наведено в ***табл. 2.***

1. При запуску програми вивести вітання та інформацію про автора (ПІБ, група).
2. Вивести в консоль поточну дату і час за допомогою спеціальних функцій.
3. Вивести в консоль: «*Формула **A:** ...*».
4. Спитати користувача, скільки елементів ряду (змінна ***J***) додати в обчисленнях.
5. Створити у програмі 3 потоки (не рахуючи головного). Назва кожного потоку повинна містити його номер.
6. Створити окремий метод для реалізації дій у потоці № 1.
7. Створити окремий загальний метод для реалізації дій у потоках № 2 і № 3.
8. Передавати номер потоку як параметр при створенні чи запуску всіх потоків.
9. В обох створених методах на початку виводити в консоль повідомлення «Потік №... почав роботу», а в кінці — «Потік №... завершив роботу» (або інші аналогічні коментарі).
10. У потоці № 1 реалізувати обчислення за формулою ***A,*** кількість ел. ряду — ***J.***
11. У потоках № 2 і 3 реалізувати обчислення цього ж ряду паралельно, загальна кількість елементів ряду — ***J*** (кожен потік додає свою половину елементів).
12. У головному потоці засікти час і запустити на виконання потік № 1.
13. Після його завершення визначити час, затрачений на послідовне обчислення. Вивести це значення в консоль (у секундах із точністю до тисячних).
14. Вивести в консоль: «*Результат послідовного обчислення: ...*»
15. У головному потоці знов засікти час і запустити на паралельне виконання потоки № 2 і № 3.
16. Після їх завершення визначити час, затрачений на паралельні обчислення. Вивести це значення в консоль (у секундах із точністю до тисячних).
17. Вивести в консоль: «*Результат паралельного обчислення: ...*»
18. Перевірити обидва отримані результати на правильність (точність) за допомогою вбудованих математичних функцій, формули обчислення суми арифметичної прогресії тощо (вивести результат їх роботи і порівняти).

19. Порівняти час, затрачений на послідовне та паралельні обчислення.
20. Визначити отримане прискорення паралельних обчислень відносно послідовного (у разах і у відсотках).
21. Згідно з результатом вивести в консоль: «*Паралельне (послідовне) обчислення є швидшим, ніж послідовне (паралельне). Прискорення = ... разів (...%)*».
22. Залежно від кількості ядер на процесорі, визначити ефективність паралельних обчислень. Вивести в консоль: «*Кількість ядер = ..., ефективність = ...*».
23. Вивести повідомлення «*Кінець головного потоку*».
24. Провести тестування програми для різної кількості ітерацій згідно з ***табл. 3*** і заповнити її (в ел. вигляді). Якщо для деяких ***J*** обчислення йдуть дуже довго (>1 хв) / дуже коротко (<0,1 с), взяти менші / більші значення ***J.***

Табл. 2. Індивідуальні варіанти до лабораторної роботи № 2

№№ вар.	***A***
1, 13, 25	$4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 ... = \pi$
2, 14, 26	$1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 ... = \pi^2 / 6$
3, 15, 27	$1/1^2 - 1/2^2 + 1/3^2 - 1/4^2 ... = \pi^2 / 12$
4, 16, 28	$1/1^2 + 1/3^2 + 1/5^2 + 1/7^2 ... = \pi^2 / 8$
5, 17, 29	$1/(1*2) + 1/(2*3) + 1/(3*4) + 1/(4*5)... = 1$
6, 18, 30	1 + 2 + 3 + 4 + 5 + 6 ...
7, 19	1 - 2 + 3 - 4 + 5 - 6 ...
8, 20	1 + 3 + 5 + 7 + 9 + 11 ...
9, 21	1 - 3 + 5 - 7 + 9 - 11 ...
10, 22	1 + 4 + 7 + 10 + 13 + 16 ...
11, 23	1 - 4 + 7 - 10 + 13 - 16 ...
12, 24	$1/1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 ... = \pi / 4$

Табл. 3. Результати тестування програми для різної кількості ітерацій

кількість ітерацій *J*	**час послідовного виконання, мс**	**час паралельного виконання, мс**	**прискорення, $S_N = T_1 / T_N$**	**ефективність, $E_N = S_N / N$**
1000				
5000				
10000				
50000				
100000				
500000				
1000000				
5000000				
10000000				

Теоретичні відомості

Для обчислення часу в сучасних мовах програмування використовуються спеціальні типи даних. У C# такий тип називається **DateTime.**

Для визначення поточного часу використовується команда **DateTime.Now.**

Приклад коду на C# для виведення поточного часу в консоль:

```
using System;
class Program
{
   static void Main()
   {
      Console.WriteLine("Current date & time: "
                        + DateTime.Now);
      Console.Read();
   }
}
```

Приклад коду на C# для задання дати та обчислення різниці між 2 датами:

```
Console.WriteLine("Current date & time: " + DateTime.Now);
DateTime a = new DateTime(2015, 10, 21, 16, 29, 0);
Console.WriteLine("Date a: " + a);
Console.WriteLine("Difference: "+(DateTime.Now - a));
Console.Read();
```

Приклад іншого конструктора дати:

```
DateTime a = new DateTime(2015, 10, 21);
```

Приклад виведення результату в днях:

```
Console.Write((DateTime.Now - a).Days);
```

Для обчислення часу в частках секунди можна використати **DateTime.Now.Ticks,** що рахує час від 1 січня 1 р. н. е. **(01.01.0001 00:00:00),** або **Environment.TickCount,** що рахує час від моменту запуску ОС (при цьому час, протягом якого система була в режимі сну, також додається до цього інтервалу).

Точність вимірювання часу визначається можливостями системного годинника. Зазвичай це 10–15 мс.

Приклад коду на C# для виведення часу роботи ОС у секундах:

```
using System;

class Program
{
    static void Main()
    {
        int t = Environment.TickCount;
        Console.WriteLine("Time since system
                startup: " + (t / 1000) + " s");
        Console.Read();
    }
}
```

Приклади створення часових міток на C# різними способами:

```
DateTime a1 = DateTime.Now;
Thread.Sleep(1000);
DateTime a2 = DateTime.Now;

long b1 = DateTime.Now.Ticks;
Thread.Sleep(1000);
long b2 = DateTime.Now.Ticks;

int c1 = Environment.TickCount;
Thread.Sleep(1000);
int c2 = Environment.TickCount;
```

Приклад визначення різниці в часі:

```
Console.WriteLine("DateTime.Now: "
                + (a2 - a1));
Console.WriteLine("DateTime.Now.Ticks: "
                + (b2 - b1));
Console.WriteLine("Environment.TickCount: "
                + (c2 - c1));
```

Приклад конвертації в інші величини без округлення самих значень:

```
(a2 - a1).TotalSeconds
(a2 - a1).TotalMilliSeconds
```

Приклад створення часових міток на C# усередині потоку:

```
static void Actions1()
  {
     Console.WriteLine("t1 starts");
     DateTime t1start = DateTime.Now;
     //Actions
     DateTime t1end = DateTime.Now;
     Console.WriteLine("time: "
       + (t1end - t1start).TotalSeconds);
     Console.WriteLine("t1 ends");
  }
```

Приклад вимірювання тривалості обчислення на 10 млн. ітерацій:

```
static void Actions1()
  {
     Console.WriteLine("t1 starts");
     double result1 = 0;
     DateTime t1start = DateTime.Now;
     for (int i = 1; i <= 10000000; i++)
     {
        result1 += Math.Pow(Math.Sin(i), 2)
                  + Math.Pow(Math.Cos(i), 2);
     }
     DateTime t1end = DateTime.Now;
     Console.WriteLine("result 1: " + result1);
     Console.WriteLine("time 1: "
           + (t1end - t1start).TotalSeconds);
     Console.WriteLine("t1 ends");
    }
```

Приклад коду для другого потоку при розбитті обчислень на 2 потоки:

```
static void Actions2()
  {
     Console.WriteLine("t2 starts");
     double result2 = 0;
     DateTime t2start = DateTime.Now;
     for (int i = 2; i <= 10000000; i+=2)
     {
        result2 += Math.Pow(Math.Sin(i), 2)
                  + Math.Pow(Math.Cos(i), 2);
     }
     DateTime t2end = DateTime.Now;
     Console.WriteLine("result 2: " + result2);
     Console.WriteLine("time 2: "
           + (t2end - t2start).TotalSeconds);
     Console.WriteLine("t2 ends");
  }
```

Приклад вимірювання загального часу роботи дочірніх потоків:

```
Thread t1 = new Thread(Actions1);
Thread t2 = new Thread(Actions2);
DateTime threads_start = DateTime.Now;
t1.Start();
t2.Start();
t1.Join();
t2.Join();
Console.WriteLine("Total time: " + (DateTime.Now - threads_start));
Console.Read();
```

При збільшенні кількості потоків можна зробити параметризований запуск. Під час запуску до потоку передається параметр типу «об'єкт» (*object*). Приклад коду:

```
Thread t1 = new Thread(Actions);
Thread t2 = new Thread(Actions);
//...
Thread t16 = new Thread(Actions);
t1.Start(1);
t2.Start(2);
//...
t16.Start(16);
```

Приклад коду загального методу для дій у потоці:

```
static int thread_count = 1;
static void Actions(object number)
  {
    int num = (int)number;
    string name = "t" + number.ToString();
    Console.WriteLine(name + " starts");
    DateTime thread_start = DateTime.Now;
    double result = 0;
    for (int i = num; i <= 10000000; i+=thread_count)
    {
       result += Math.Pow(Math.Sin(i), 2)
              + Math.Pow(Math.Cos(i), 2);
    }
    DateTime thread_end = DateTime.Now;
    Console.WriteLine(name + " result: " + result);
    Console.WriteLine(name + " time: "
           + (thread_end - thread_start).TotalSeconds);
        Console.WriteLine(name + " ends");
  }
```

Основні показники якості паралельної програми — це **прискорення** та **ефективність.**

Прискорення обчислюється як:

$$S_N = \frac{T_1}{T_N}, \tag{1}$$

де $\boldsymbol{T_1}$ – час на одному процесорі,
$\boldsymbol{T_N}$ – час на $\boldsymbol{N}$ процесорах.
Ефективність обчислюється як:

$$E_N = \frac{S_N}{N} = \frac{T_1}{NT_N}, \tag{2}$$

Важливо розуміти, що прискорення і ефективність — не тотожні. Збільшуючи ***N,*** ми збільшуємо прискорення ***S.*** Однак ефективність може рости не так швидко.

Ідеальне прискорення: $\boldsymbol{S = N.}$

Однак ідеальна ефективність $\boldsymbol{E} = 1$ досягається при $\boldsymbol{N} = 1$.

Вимірявши час, витрачений на ті самі обчислення послідовною та паралельною програмою, можна підставити його у формули (1) і (2) та обчислити відповідно прискорення та ефективність паралельної програми.

При цьому залежно від кількості ітерацій значення ***S*** та ***E*** можуть змінюватись, тому для об’єктивної картини слід провести серію експериментів із різною кількістю ітерацій.

Висновки

У ході виконання лабораторної роботи визначено, яким чином можна обчислити час виконання, а також розрахувати прискорення та ефективність роботи паралельної програми порівняно з послідовною.

Контрольні запитання

1. Чи доцільно використовувати паралельні обчислення для поставленої задачі?
2. Від чого залежить прискорення паралельних обчислень?
3. Від чого залежить ефективність паралельних обчислень?
4. Чим відрізняються показники «прискорення» та «ефективність»?
5. Що таке параметризований запуск потоку?

Література

Див. джерела №№ 1, 3, 4, 5.

ЛАБОРАТОРНА РОБОТА 3.

Взаємодія з користувачем у консольній програмі

(введення та виведення даних)

Мета: навчитися реалізовувати взаємодію з користувачем у консольній програмі при введенні й виведенні даних, а також забезпечувати валідацію даних.

Завдання

Написати програму довільною мовою програмування (C#, Python, Java, C++, інші) для введення користувачем **текстових і числових** даних у **консоль,** обробки цих даних і виведення результатів обробки в **консоль.**

Призначення, функції програми та їх кількість, послідовність дій — **довільні** (чим оригінальніше — тим краще).

Приклади можливих функцій у програмі: визначення віку користувача за датою народження; обчислення певної математичної або економічної формули; конвертація між різними системами вимірювання (км / милі, °C / °F і т.д.); транслітерація імені; короткий діалог із користувачем; психологічний тест тощо.

Програма повинна:

1. Мати простий та зрозумілий консольний інтерфейс для взаємодії з користувачем.
2. При запуску виводити вітання та інформацію про автора (ім'я та прізвище, № групи).
3. Підтримувати введення і виведення літер латинкою (англійська розкладка клавіатури) та кирилицею (українська розкладка).
4. Підтримувати введення і виведення цифр, пробілів, знаків пунктуації, спеціальних символів.
5. Фільтрувати некоректне введення даних (тип даних, заданий розмір тощо) користувачем (блокувати таке введення; вимагати повторного введення і т.п.).
6. Виводити на екран коректну інформацію (точну копію введених даних без спотворень і/або результати обчислень над даними чи іншої їх обробки тощо).
7. Не зависати і не завершуватись аварійно при будь-яких діях користувача всередині програми.
8. Давати можливість вводити дані повторно без перезапуску цілої програми (не завершуватись автоматично після 1 циклу введення-виведення і т.д.).
9. Завершувати свою роботу лише за згодою користувача.

Теоретичні відомості

Інтерфейс — сполучна ланка між елементами комп'ютерної системи. Це поняття включає в себе засоби та правила взаємодії між елементами.

Серед видів інтерфейсів виділяють:

- фізичний (апаратний) інтерфейс (роз'єми, кабелі);
- програмний інтерфейс:
    - інтерфейс (протокол, абстрактний клас) у ООП;

    - application programming interface (API);
    - application binary interface (ABI);
    - інтерфейс користувача.
- мережевий (програмний і/або апаратний).

Серед інтерфейсів користувача можна виділити:

- текстовий (консольний, command-line);
- графічний (GUI – graphical user interface):
    - віконний;
    - веб-інтерфейс (сторінковий);
- сенсорний;
- голосовий;
- жестовий;
- інші.

При створенні інтерфейсу користувача слід враховувати:

- **специфіку галузі застосування (або інтереси замовника):**
    - досягнення цілей розробки;
    - збільшення кількості користувачів.
- **очікування користувачів системи:**
    - реалізація необхідних функцій;
    - зручність використання.
- **наявні обмеження та проблеми:**
    - коректність даних;
    - обсяги даних;
    - безпека даних.

Основними вимогами до інтерфейсу користувача є:

- ефективне виконання завдань;
- відповідність вимогам замовника / користувача (а не розробника);
- логічність дій (принцип *DWIM – do what I mean*);
- підтвердження користувачем важливих дій (виходу з програми, перезапису файлів тощо);
- захист від помилок;
- робота без затримок (без зависання);
- коментарі про хід виконання дій.

Захист від помилок передбачає коректне функціонування програми навіть при некоректних діях користувача.

Шляхи захисту від помилок можуть бути наступні.

Для запобігання помилкам:

- **визначення обмежень** на введені дані:
    - тип даних;
    - допустимі значення (інтервал, перелік);
- **доведення вимог до відома користувача:**
    - пояснення;
    - приклад формату введення;
- **фільтрація** введених даних:

    - посимвольна або після введення рядка;
    - за кодом клавіші або символом;
- **блокування** кнопок та інших елементів;
- **налаштування** кодування тексту, мови, регіональних стандартів.

Для опрацювання помилок, які вже стались:

- визначення причин помилки;
- видача докладного повідомлення:
    - суть помилки;
    - місце помилки;
    - шляхи виправлення;
- можливість виправлення / повторного введення даних користувачем.

Висновки

У ході виконання лабораторної роботи визначено, що валідація даних і опрацювання помилок при реалізації інтерфейсу користувача є необхідними для коректної роботи програми.

Контрольні запитання

1. Що таке змінна?
2. Які властивості має змінна?
3. Які бувають змінні?
4. Що таке потік введення і потік виведення (input & output stream)?
5. Як працюють потоки введення та виведення?

Література

Див. джерело № 2.

ЛАБОРАТОРНА РОБОТА 4.

Багатопоточна програма з графічним інтерфейсом користувача

Мета: навчитися створювати багатопоточні програми з графічним інтерфейсом користувача та забезпечувати захист програми від зависання за рахунок роботи та взаємодії паралельних потоків.

Завдання

Написати програму з графічним інтерфейсом користувача довільною мовою програмування (C#, Python, Java, C++, інші) з використанням потоків для паралельних обчислень.

Формулу **A** підставити у завдання відповідно до свого варіанту (він визначається як порядковий номер студента в загальному списку групи). Значення по варіантах наведено в ***табл. 4.***

1. Встановити заголовок головного вікна: «Лабораторна робота №4, 20__ рік».
2. Створити у вікні програми дві вкладки: «*Програма*», «*Про автора*».
3. У «*Про автора*» вивести інформацію про себе (ПІБ, група) в елемент Label (або аналогічний). Шрифт — Arial, 16 пт., жирний, вирівн. по центру форми.
4. У вкладці «*Програма*» вивести напис: «*Формула **A:** ...*» і свою формулу.
5. Створити напис «*Кількість ітерацій **J:***» і числове поле (NumericUpDown або аналогічне) для введення користувачем числа ***J.*** Мінімальне значення — 2. Значення за замовчуванням — 1 млн.
6. Створити напис «*Кількість потоків:*» і випадаючий список без можливості ручного введення (лише вибір зі списку). У списку має бути вибір кількості потоків ***T:*** 2, 4, 8, 16.
7. Створити кнопку «*Запустити паралельні обчислення*». При наведенні курсора на кнопку має змінюватись колір її фону та шрифту.
8. Створити 3 радіокнопки з назвами кольорів. При виборі має змінюватись колір фону вкладки «*Програма*». Якщо не обрано, колір має лишатись незмінним.
9. Створити багаторядкове текстове поле для виведення коментарів про дії.
10. Створити один загальний метод для реалізації дій у паралельних потоках:
    - Передавати номер потоку в якості параметра.
    - Реалізувати обчислення за формулою ***A,*** загальна кількість елементів ряду — ***J*** (кожен потік виконує рівну частину ітерацій).
11. Створити обробник події натиснення на кнопку:
    - Вивести в текстове поле: «*Початок паралельних обчислень. Кількість потоків: ... Загальна кількість ітерацій: ...*»
    - Створити задану користувачем кількість потоків ***T.***
    - Засікти час і запустити потоки на паралельне виконання.
    - Під час обчислень тимчасово блокувати кнопку.
    - Після їх завершення визначити час, затрачений на паралельні обчислення.
    - Вивести: «*Обчислення завершено. Затрачений час: ... Результат: ...*»

- Перевірити результат на правильність (точність) за допомогою вбудованих математичних функцій, формули обчислення суми арифметичної прогресії тощо (вивести результат їх роботи в текстове поле і порівняти).

12. При повторних обчисленнях не стирати, а доповнювати вміст текстового поля.

13. Автоматично прокручувати вміст текстового поля до останнього результату.

14. Інтерфейс програми не повинен зависати при виконанні обчислень! Під зависанням розуміється затримка у відгуку на дії користувача довше 0,5–1 с.

Табл. 4. Індивідуальні варіанти до лабораторної роботи № 4

№№ вар.	***A***
1, 13, 25	$4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 ... = \pi$
2, 14, 26	$1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 ... = \pi^2 / 6$
3, 15, 27	$1/1^2 - 1/2^2 + 1/3^2 - 1/4^2 ... = \pi^2 / 12$
4, 16, 28	$1/1^2 + 1/3^2 + 1/5^2 + 1/7^2 ... = \pi^2 / 8$
5, 17, 29	1/(1*2) + 1/(2*3) + 1/(3*4) + 1/(4*5)... = 1
6, 18, 30	1 + 2 + 3 + 4 + 5 + 6 ...
7, 19	1 - 2 + 3 - 4 + 5 - 6 ...
8, 20	1 + 3 + 5 + 7 + 9 + 11 ...
9, 21	1 - 3 + 5 - 7 + 9 - 11 ...
10, 22	1 + 4 + 7 + 10 + 13 + 16 ...
11, 23	1 - 4 + 7 - 10 + 13 - 16 ...
12, 24	$1/1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 ... = \pi / 4$

Теоретичні відомості

При створенні графічного інтерфейсу користувача (англ. *GUI — graphical user interface*) використовуються стандартні візуальні елементи (назви на прикладі реалізації в C#):

- **вікна** (форми) — **`Form`**
- **кнопки** — **`Button`**
- **написи** (підписи) — **`Label`**
- **поля введення:**
  - текстове поле — **`TextBox`**
  - розширене текстове поле — **`RichTextBox`**
- **елементи вибору:**
  - спадний список — **`ComboBox`**
  - числове поле — **`NumericUpDown`**
  - прапорець (галочка) — **`CheckBox`**
  - перемикач (радіокнопка) — **`RadioButton`**
- **вкладки** — **`TabControl`**
- **таблиці** — **`DataGridView`**
- **повідомлення** — **`MessageBox`**

Етапи роботи з елементами GUI у середовищі Microsoft Visual Studio:

1. Конструктор (*Design Time*). Це створення та налаштування елементів за допомогою візуального редактора. Дизайн, створений таким чином, показується на екрані початково при запуску програми.

2. Виконання (*Run Time*). Передбачає створення та налаштування елементів безпосередньо в коді. Дозволяє реалізувати зміни під час виконання програми

Для різних елементів можна налаштувати деякі загальні властивості:

- *.Name*
- *.Size, .Height, .Width*
- *.Location, .Left, .Top*
- *.ForeColor, .BackColor*
- *.Font → .Name, .Size, .TextAlign*
- *.Enabled (true / false)*
- *.Visible (true / false)*
- *.ReadOnly (true / false)*
- *.TabIndex (0, 1, 2, ...)*

Інші властивості відрізняються залежно від конкретного елемента GUI. Всі їх можна налаштувати в середовищі розроблення у вкладці *Properties* (Властивості).

При реалізації текстових полів для введення даних користувачами слід проводити валідацію. Зокрема доцільно реалізувати наступні перевірки:

- наявність даних;
- тип даних (введені символи: цифри, літери, цілі / дробові числа тощо);
- формат введення (кома / крапка, верхній / нижній регістр і т.д.);
- допустимі значення:
  - перелік (‘Y’, ‘N’);
  - граничні значення (min..max; A..Z);
  - винятки (0; Esc);
- коректність відносно ін. значень (y>x, рядок a довше за b тощо).

При виникненні помилок у ході виконання програми можна сповіщати про них, наприклад, шляхом показу спливаючих повідомлень (у C# за це відповідає елемент ***MessageBox***). При цьому слід реалізувати наступне:

- для різних помилок повинні бути **різні повідомлення:**
  - відсутність даних;
  - неправильний тип;
  - неправильний формат;
  - вихід за допустимі межі;
  - несумісність із іншими даними;
  - проблеми при записі у файл;
  - інше;
- визначати **докладне місце помилки;**
- очікування **повторного введення.**

Для запобігання зависанню GUI слід використовувати багатопоточність, реалізувавши обчислення та всі інші дії в окремих від інтерфейсу паралельних потоках. Таким чином головний потік (*Main*), який відповідає за GUI, не буде

залежати від інших дій і не муситиме очікувати на події, що може спричиняти зависання програми.

Загальні рекомендації для протидії зависанню програми наступні:

1. Не слід перевантажувати головний потік (*Main*):

- винести в інші потоки все, що можна (наприклад, обчислення, роботу з файлами тощо);
- решту операцій (наприклад, виведення на екран) скоротити.

2. Не можна блокувати головний потік:

- методи **`Join(), Sleep()`** та подібні (які вимагають очікування та призупинки роботи потоку, в якому їх викликано) слід використовувати лише в побічних потоках.

Якщо є потреба передавати дані з побічних потоків у головний (наприклад, для виведення результатів обчислень на екран), то для коректної взаємодії з GUI з інших потоків можна використовувати наступні механізми:

1) делегати;
2) опрацювання подій;
3) вбудовані можливості конкретної мови програмування (наприклад, клас ***BackgroundWorker*** у C#).

Загальний алгоритм роботи з ***BackgroundWorker:***

1. Поміщаємо на форму елемент **`BackgroundWorker`** (наприклад, bgw1).
2. Описуємо дії для виконання в побічному потоці через подію **`DoWork.`**
3. Описуємо дії після виконання потоку через подію **`RunWorkerCompleted.`**
4. Запускаємо новий потік на виконання методом **`bgw1.RunWorkerAsync().`**

Висновки

У ході виконання лабораторної роботи визначено, що при реалізації програм із графічним інтерфейсом користувача можна забезпечити захист від зависання за рахунок роботи та взаємодії паралельних потоків.

Контрольні запитання

1. Що таке інтерфейс? Види інтерфейсів.
2. Що слід враховувати при проектуванні інтерфейсу користувача?
3. Як можна запобігти помилкам при проектуванні графічного інтерфейсу?
4. Як слід обробляти помилки, які вже сталися?
5. Основні вимоги до зовнішнього вигляду графічного інтерфейсу.
6. Для чого потрібна багатопоточність при реалізації інтерфейсу користувача?
7. Чи дає ефект багатопоточність на одноядерному процесорі?

Література

Див. джерела №№ 2, 3, 5.

ЛАБОРАТОРНА РОБОТА 5.

Багатопоточна програма з розширеним графічним інтерфейсом

Мета: навчитися створювати багатопоточні програми з розширеним графічним інтерфейсом користувача для керування обчисленнями в паралельних потоках.

Завдання

Написати програму з графічним інтерфейсом користувача довільною мовою програмування (C#, Python, Java, C++, інші) з використанням потоків для паралельних обчислень.

Значення ***A, B, C*** підставити у завдання відповідно до свого варіанту (він визначається як порядковий номер студента в загальному списку групи). Значення по варіантах наведено в ***табл. 5.***

1. Встановити заголовок головного вікна: «*ЛР № 5, автор: ...*» (прізвище, група).
2. Створити на формі напис: «*Формула* ***A:*** *...*» і свою формулу.
3. Створити напис «*Кількість ітерацій* ***J:***» і числове поле (NumericUpDown або аналогічне) для ручного введення користувачем числа ***J.*** Мінімальне значення — 100. Значення за замовчуванням — 10 тис.
4. Створити напис «*Кількість потоків* ***T:***» і числове поле з можливістю ручного введення. Дати можливість користувачу ввести значення ***T*** від 1 до 100.
5. Для обох числових полів реалізувати захист від некоректного введення.
6. Створити кнопку «*Запустити паралельні обчислення*».
7. Створити багаторядкове текстове поле для виведення коментарів про дії.
8. Створити один загальний метод для реалізації дій у паралельних потоках:
   - Реалізувати обчислення за формулою ***A,*** загальна кількість ітерацій — ***J*** (кожен потік виконує рівну частину ітерацій).
   - Після кожної ітерації потік має «засинати» на ***B*** мілісекунд.
9. Створити обробник події натиснення на кнопку:
   - Вивести в текстове поле: «*Початок паралельних обчислень. Кількість потоків: ... Загальна кількість ітерацій: ...*».
   - Створити задану користувачем кількість потоків ***T.***
   - Вивести: «*Обчислення завершено. Результат: ...*».
   - Перевірити результат на правильність (точність) за допомогою вбудованих математичних функцій, формули обчислення суми арифметичної прогресії тощо (вивести результат їх роботи в текстове поле і порівняти).
10. При повторних обчисленнях не стирати, а доповнювати вміст текстового поля.
11. Автоматично прокручувати вміст текстового поля до останнього результату.
12. Створити кнопку, яка повністю скасовує обчислення.
13. Створити кнопку, яка тимчасово призупиняє обчислення.
14. Створити кнопку, яка відновлює обчислення після зупинки.
15. Блокувати кнопки тоді, коли їхнє натиснення може призвести до помилок.

16. Створити на формі індикатор виконання (ProgressBar або аналогічний), який смугою кольору ***C*** показує хід виконання обчислень із точністю до 1%.
17. При закритті головного вікна програми всі виконувані потоки мають припиняти свою роботу.
18. Інтерфейс програми не повинен зависати при виконанні дій! Під зависанням розуміється затримка у відгуку на дії користувача довше 0,5–1 с.
19. Не використовувати методи ***Join*** і ***Sleep*** у головному потоці вашої програми.
20. Програма не повинна ламатись при зупинці та відновленні роботи

Табл. 5. Індивідуальні варіанти до лабораторної роботи № 5

№ вар.	*A*	*B*	*C*
1	$4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 ... = \pi$	16	зелений
2	$1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 ... = \pi^2 / 6$	17	синій
3	$1/1^2 - 1/2^2 + 1/3^2 - 1/4^2 ... = \pi^2 / 12$	18	жовтий
4	$1/1^2 + 1/3^2 + 1/5^2 + 1/7^2 ... = \pi^2 / 8$	19	червоний
5	$1/(1*2) + 1/(2*3) + 1/(3*4) + 1/(4*5)... = 1$	10	чорний
6	1 + 2 + 3 + 4 + 5 + 6 ...	11	помаранчевий
7	1 - 2 + 3 - 4 + 5 - 6 ...	12	фіолетовий
8	1 + 3 + 5 + 7 + 9 + 11 ...	13	коричневий
9	1 - 3 + 5 - 7 + 9 - 11 ...	14	темно-синій
10	1 + 4 + 7 + 10 + 13 + 16 ...	15	світло-жовтий
11	1 - 4 + 7 - 10 + 13 - 16 ...	16	темно-червоний
12	$1/1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 ... = \pi / 4$	17	темно-сірий
13	$4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 ... = \pi$	18	світло-сірий
14	$1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 ... = \pi^2 / 6$	19	темно-зелений
15	$1/1^2 - 1/2^2 + 1/3^2 - 1/4^2 ... = \pi^2 / 12$	10	салатовий
16	$1/1^2 + 1/3^2 + 1/5^2 + 1/7^2 ... = \pi^2 / 8$	11	бірюзовий
17	$1/(1*2) + 1/(2*3) + 1/(3*4) + 1/(4*5)... = 1$	12	блакитний
18	1 + 2 + 3 + 4 + 5 + 6 ...	13	темно-коричневий
19	1 - 2 + 3 - 4 + 5 - 6 ...	14	світло-коричневий
20	1 + 3 + 5 + 7 + 9 + 11 ...	15	світло-помаранчевий
21	1 - 3 + 5 - 7 + 9 - 11 ...	16	бузковий
22	1 + 4 + 7 + 10 + 13 + 16 ...	17	рожевий
23	1 - 4 + 7 - 10 + 13 - 16 ...	18	бежевий
24	$1/1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 ... = \pi / 4$	19	білий
25	$4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 ... = \pi$	10	оливковий
26	$1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 ... = \pi^2 / 6$	11	світло-фіолетовий
27	$1/1^2 - 1/2^2 + 1/3^2 - 1/4^2 ... = \pi^2 / 12$	12	темно-фіолетовий
28	$1/1^2 + 1/3^2 + 1/5^2 + 1/7^2 ... = \pi^2 / 8$	13	темно-бірюзовий
29	$1/(1*2) + 1/(2*3) + 1/(3*4) + 1/(4*5)... = 1$	14	світло-зелений
30	1 + 2 + 3 + 4 + 5 + 6 ...	15	сірий

Теоретичні відомості

Програми можуть зависати (довго не відповідати) не лише через штучні перешкоди в роботі головного потоку (наприклад, виклик **`Join(), Sleep()`** або перевантаження обчисленнями й іншими діями), а й із природних причин. Якщо поставлене завдання саме по собі займає забагато часу (тривалі обчислення, робота з об'ємними файлами тощо), це може затримувати роботу всієї програми в цілому. В таких випадках може бути доцільно:

1. Дати змогу користувачам перервати (тимчасово призупинити чи повністю зупинити) тривалий процес.

2. Коментувати хід виконання програми (що відбувається, який % дій уже виконано, скільки ще часу приблизно потрібно для їх завершення).

Програмно тимчасове призупинення потоків можна реалізувати на C# так:

```
t1.Suspend();
...
t1.Resume();
```

Тим не менше, ці методи можуть бути не рекомендованими для використання, оскільки блокування потоку іноді може викликати блокування пам'яті, з якою він працював. Альтернативою є використання подій та їхніх обробників.

Для повного скасування дій, що виконуються в потоці ***BackgroundWorker,*** можна викликати метод **`bgw1.CancelAsync()`**

Для відстеження ходу виконання дій у ***BackgroundWorker*** можна скористатись властивістю **`bgw1.WorkerReportsProgress:`**

- встановити її значення = **`true`**
- використовувати подію **`ProgressChanged`**
- а також властивість **`ProgressPercentage`**

Для візуального відображення ходу виконання дій у програмі можна помістити на екран елемент **`ProgressBar.`** У ньому можна налаштувати колір, прозорість та інші ефекти.

При цьому слід пам'ятати, що налаштування графіки в ОС можуть впливати на відображення елементів, заміняючи встановлені кольори на стандартні для обраної користувачем кольорової гами (стилів, теми) в ОС. У такому разі треба додатково перевизначити в коді, аби необхідні нам властивості перекривали вбудовані налаштування ОС.

Висновки

У ході виконання лабораторної роботи визначено, що при створенні багатопоточних програм із графічним інтерфейсом користувача можна керувати обчисленнями в паралельних потоках шляхом їх призупинення та відновлення,

повної зупинки тощо. Це реалізовується завдяки відповідним візуальним елементам графічного інтерфейсу.

Контрольні запитання

1. Захист від некоректного введення у текстові та числові поля.
2. Блокування кнопок на формі.
3. Методи протидії зависанню графічного інтерфейсу програми.
4. Чому не слід використовувати методи Join і Sleep у головному потоці?
5. Захист від помилок при доступі побічних потоків до елементів інтерфейсу.
6. Яким чином можна реалізувати повну зупинку виконання потоку?
7. Варіанти реалізації тимчасового призупинення та відновлення роботи потоку.
8. Індикатор виконання (progress bar) та його особливості.

Література

Див. джерело № 3.

ЛАБОРАТОРНА РОБОТА 6.

Синхронізація потоків у багатопоточній програмі

Мета: навчитися створювати багатопоточні програми із синхронізацією потоків для реалізації паралельних обчислень.

Завдання

Написати програму з графічним інтерфейсом користувача довільною мовою програмування (C#, Python, Java, C++, інші) з використанням потоків для генерації, сортування та запису великого масиву чисел у текстові файли.

Значення ***A*** і ***B*** підставити у завдання відповідно до свого варіанту (він визначається як порядковий номер студента в загальному списку групи). Значення по варіантах наведенj в ***табл. 6.***

Алгоритми видів сортування та їхні готові реалізації для різних мов програмування можна знайти в мережі інтернет і використати в роботі.

Максимальний бал за лабораторну залежить від способу реалізації **п. 8.**

1. Встановити заголовок головного вікна: «*ЛР № 6, автор: ...*» (прізвище, група).
2. Створити на формі напис: «*Генерація чисел у межах: ...*» і свій діапазон (***A***).
3. Створити напис «*Довжина масиву* ***M:***» і текстове поле (TextBox чи аналогічне) для ручного введення ***M.*** Мінімальне значення — 10 тис. Значення за замовчуванням — 1 млн. Реалізувати захист від некоректного введення.
4. Створити кнопку «*Запуск*».
5. Створити багаторядкове текстове поле для виведення коментарів про дії.
6. Створити окремі потоки та методи (мінімум по одному) для наступних дій:
   - генерація чисел;
   - сортування чисел.
7. Створити обробник події натиснення на кнопку:
   - Вивести в поле коментарів: «*Початок генерації. Довжина масиву: ...*».
   - Створити масив чисел довжиною ***M.***
   - Заповнити масив випадковими числами, кожне число в межах діапазону ***A.***
   - Записати створений масив у файл **generated.txt** у довільній папці.
   - Вивести в поле коментарів: «*Генерацію завершено*».
   - Автоматично відкрити папку з файлом і самий файл для перегляду.
   - Відсортувати створений масив чисел за зростанням за методом ***B.***
   - Записати відсортований масив у файл **sorted.txt** у тій же папці.
   - Вивести в поле коментарів: «*Сортування завершено*».
   - Автоматично відкрити цей файл для перегляду (в *Блокноті* чи ін. програмі).
8. Узгодити взаємодію потоку, який генерує числа, і потоку, який сортує їх, обравши на власний розсуд спосіб їх синхронізації:

- **<u>на максимальний бал</u>:** через <u>один із</u> методів синхронізації, розглянутих на лекціях (для C#: ***lock, Wait/Pulse, mutex, семафори, події, передача повідомлень;*** для інших мов програмування — один із їхніх відповідників);
- **<u>на 80% оцінки</u>:** без використання цих методів (наприклад, через ***Join*** і т.д.).

9. За необхідності також розділити генерацію і/або сортування на декілька потоків для прискорення обчислень.

10. При записі у файли використовувати розділювачі між числами (пробіл, кома, табуляція і т.д. — на свій вибір).

11. Записувати числа у файли великими блоками, а не по одному числу.

12. При повторному запуску обчислень стирати попередній вміст поля коментарів.

13. Створити кнопку, яка скасовує обчислення. При натисненні вивести коментар «*Обчислення скасовано*» у діалоговому вікні (MessageBox або аналогічне). Також додати рядок «*Обчислення скасовано.*» до поля коментарів.

14. Блокувати кнопки тоді, коли їхнє натиснення може призвести до помилок.

15. При закритті головного вікна програми всі виконувані потоки мають припиняти свою роботу.

16. Інтерфейс програми не повинен зависати при виконанні дій! Під зависанням розуміється затримка у відгуку на дії користувача довше 0,5–1 с.

17. Не використовувати методи ***Join*** і ***Sleep*** у головному потоці вашої програми.

Табл. 6. Індивідуальні варіанти до лабораторної роботи № 6

№ вар.	***A***	***B****
1	1..1000	Selection Sort
2	2..2000	Bubble Sort
3	3..3000	Cocktail Sort
4	4..4000	Gnome Sort
5	5..5000	Insertion Sort
6	6..6000	Merge Sort
7	7..7000	Tree Sort
8	8..8000	Timsort
9	9..9000	Counting Sort
10	10..10000	Bucket Sort
11	11..11000	Shell Sort
12	12..12000	Comb Sort
13	13..13000	Heap Sort
14	14..14000	Smooth Sort
15	15..15000	Quicksort
16	16..16000	Introsort
17	17..17000	Patience Sorting
18	18..18000	Stooge Sort
19	19..19000	Radix Sort
20	20..20000	Selection Sort

21	21..21000	Bubble Sort
22	22..22000	Cocktail Sort
23	23..23000	Gnome Sort
24	24..24000	Insertion Sort
25	25..25000	Merge Sort
26	26..26000	Tree Sort
27	27..27000	Timsort
28	28..28000	Counting Sort
29	29..29000	Bucket Sort
30	30..30000	Shell Sort

Теоретичні відомості

Критичний ресурс — ресурс, який допускає обслуговування лише 1 користувача за 1 раз. Серед таких ресурсів:

- процесор;
- оперативна пам’ять;
- постійна пам’ять;
- пристрої введення / виведення.

Критична ділянка (секція) — місце програми, в якому відбувається звернення до критичних ресурсів.

Синхронізація — узгодження взаємодії процесів (потоків):

- при зверненні до критичних ресурсів (у кожен момент часу критичний ресурс може використовуватись лише одним процесом / потоком);
- при очікуванні певних подій (коли один із процесів / потоків чекає на настання певної події в іншому процесі / потоці).

Можливі методи синхронізації потоків:

1. Блокування (замок).
2. Wait(), Pulse(), PulseAll().
3. Взаємовиключення.
4. Семафори.
5. Події.
6. Передача повідомлень.

1. У першому методі синхронізації використовується об’єкт **`lock`** (замок) Замок керує доступом до блоку коду.

При блокуванні:

- замкнута ділянка коду може бути використана лише потоком, який тримає замок;
- усі інші потоки блокуються (режим очікування);
- при виході з ділянки замок відмикається.

Це дозволяє запобігати конфліктам при доступі до критичних ресурсів.

Приклад коду на C# для реалізації блокування (замка):

```
static object obj = new object();
//...
for (int i = 1; i <= 1000000; i++)
    {
       lock (obj)
        {
            sum++;    //критична ділянка
        }
    }
```

2. Методи **`Wait(), Pulse(), PulseAll()`** використовуються, коли потік, який тримає замок, очікує на інший ресурс.

Метод **`Pulse()`** (надіслати імпульс) дозволяє виконання іншому потоку, який чекає на цей замок (першому в черзі).

Метод **`PulseAll()`** дозволяє виконання всім іншим потокам, які чекають на цей замок.

3. У методі **взаємовиключень** використовується об'єкт **`mutex`** (від англ. ***mut****ual* ***ex****clusion* — взаємовиключення). Цей механізм запропоновано в 1965 р. Едсгером Дейкстрою. При взаємовиключеннях лише 1 потік може мати доступ до критичної ділянки.

М'ютекс подібний до замка, проте він:

- може бути доступним із різних процесів у рамках цілої ОС;
- повільніший, ніж замок.

Через це доцільно використовувати:

- **`mutex`** — для загальних ресурсів у ОС;
- **`lock`** — для змінних і т.д.

Методи для роботи з м'ютексами:

- **`WaitOne()`**
  - чекає, поки м'ютекс звільниться;
  - блокує викликаючий потік;
- **`ReleaseMutex()`**
  - вивільняє м'ютекс;
  - дозволяє іншим роботу з ним.

4. Семафори визначають максимальну кількість потоків, які можуть мати доступ до ресурсу. Семафори використовуються тоді, коли немає значення, якому потоку дозволити використання ресурсу.

Семафор містить **лічильник,** значення якого = кількість дозволів. При наданні 1 дозволу значення лічильника зменшується, і навпаки.

Види семафорів:

- двійковий (0, 1)
- трійковий (0, 1, 2)
- ...

М’ютекс і замок по суті є частковими випадками семафору (двійкового). Однак м’ютекс прив’язаний до конкретного потоку, а семафор — ні. Семафор знаходиться за межами потоків і керує ними зовні.

Основні дії над семафорами:

- ініціалізація;
- перевірка стану;
- збільшення / зменшення значення.

Приклад коду для ініціалізації семафора в C#:

```
Semaphore sem = new Semaphore(2,2);
```

Приклад коду на C# для роботи з семафором:

```
WaitOne()
//дії всередині семафора
Release()
```

5. Події використовуються, коли один потік чекає на подію, яка має відбутися в іншому потоці.

Послідовність опрацювання подій:

- Потік, що чекає, викликає **`WaitOne():`**
  - якщо об’єкт знаходиться в потрібному стані, метод повертається негайно;
  - в іншому випадку викликаючий потік призупиняється до отримання сигналу.
- Метод **`Set()`** подає сигнал про подію.
- **`Reset()`** повертає подію в поч. стан.

Для повернення подій у початковий стан використовують:

- **`ManualResetEvent`** — для повернення вручну;
- **`AutoResetEvent`** — автоматично після передачі сигналу очікуючому потоку.

6. Останній із розглянутих методів (*message passing*) передбачає, що при паралельній роботі потоків між ними передаються **повідомлення.** Вони містять дані, важливі для взаємодії потоків.

Механізм передачі повідомлень наступний:

- при критичних операціях один потік створює повідомлення;
- повідомлення надходять у чергу;
- інший потік зчитує повідомлення з черги.

Передача повідомлень найчастіше використовується для взаємодії процесів, однак іноді може застосовуватись і для потоків.

Висновки

У ході виконання лабораторної роботи визначено, що синхронізація роботи потоків у паралельній програмі дозволяє узгодити роботу потоків і запобігти помилкам при обчисленнях, забезпечивши реалізацію логіки щодо необхідної послідовності виконання програми.

Контрольні запитання

1. Методи синхронізації потоків та їхні особливості.
2. Чи можна / доцільно ділити на кілька потоків генерацію та сортування чисел?
3. Особливості роботи з текстовими файлами:
   - відкриття / закриття;
   - читання / запис;
   - захист від помилок.
4. Чим відрізняються потік і процес?
5. Чи є сенс ділити процес на потоки замість створення багатопроцесних програм?

Література

Див. джерела №№ 1, 3, 4.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Базова

1. **Погорілий С. Д.** Методи кластерних обчислень : моногр. / С. Д. Погорілий, Ю. В. Бойко, Р. І. Левченко, В. А. Мар'яновський ; Київ. нац. ун-т ім. Т. Шевченка. – К.: Київ. ун-т, 2013. – 415 с. – ISBN 978-966-439-679-1.
2. **Татарчук М. І.** Корпоративні інформаційні системи : навч. посіб. / Київ. нац. екон. ун-т. – К. : КНЕУ, 2005. – 291 с. – ISBN 966-574-717-7.

Допоміжна

3. **Семеренко В. П.** Технології паралельних обчислень : навч. посіб. – Вінниця: ВНТУ, 2018. – 104 с.
4. **Старченко В. В.** Паралельне програмування: метод. рекомендації до виконання практичних та самостійних робіт. – Миколаїв: ЧНУ ім. Петра Могили, 2021. – 40 с.
5. **Костіков М. П.** Можливості Python для паралельного програмування // Наук. пр. IV міжнар. наук.-практ. конф. «Сучасні тенденції розвитку інформаційних систем і телекомунікаційних технологій», 1–2 лют. 2022 р. (Київ, Україна). – К.: НУХТ, 2022. – С. 89–90.
6. **Костіков М. П.** Побудова розподілених систем інтернету речей із використанням SQLite // Матер. VII Міжнар. наук.-техн. Internet-конф. «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами», 26 листоп. 2020 р. – К.: НУХТ, 2020. – С. 239.

Інформаційні ресурси

1. **C# Corner.** Community of Software & Data Developers [online]. URL: https://www.c-sharpcorner.com.
2. **Python Software Foundation.** Welcome to Python.org [online]. URL: https://www.python.org.
3. **Oracle Corporation.** Java [online]. URL: https://www.java.com.
4. CPlusPlus [online]. URL: https://www.cplusplus.com.