МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до проведення практичних занять та організації самостійної роботи
з навчальної дисципліни

«ТЕХНОЛОГІЇ РОЗПОДІЛЕНИХ СИСТЕМ ТА ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ»

(для здобувачів першого (бакалаврського) рівня вищої освіти денної та заочної форм навчання зі спеціальності 122 – Комп'ютерні науки, освітньо-професійна програма «Комп'ютерні науки»)

Харків
ХНУМГ ім. О. М. Бекетова
2023

Методичні рекомендації до проведення практичних занять та організації самостійної роботи з навчальної дисципліни «Технології розподілених систем та паралельних обчислень» (для здобувачів першого (бакалаврського) рівня вищої освіти денної та заочної форм навчання зі спеціальності 122 – Комп'ютерні науки, освітньо-професійна програма «Комп'ютерні науки» / Харків. нац. ун-т міськ. госп. ім. О. М. Бекетова ; уклад. К. В. Плахотніков. – Харків : ХНУМГ ім. О. М. Бекетова, 2023. – 43с.

Укладач канд. техн. наук, доц. К. В. Плахотніков

Рецензент

Н. Д. Сізова, доктор фізико-математичних наук, професор, професор кафедри комп'ютерних наук та інформаційних технологій Харківського національного університету міського господарства імені О. М. Бекетова

Рекомендовано кафедрою комп'ютерних наук та інформаційних технологій, протокол № 9 від 27 січня 2023 р.

Методичні рекомендації призначені для здобувачів спеціальності 122 – Комп'ютерні науки. Подано засоби та послідовність виконання завдань, запитання для самопідготовки, список рекомендованих джерел тощо.

ЗМІСТ

ВСТУП

Метою навчальної дисципліни «Технології розподілених систем та паралельних обчислень» є надання теоретичних та практичних знань з побудови складних високопродуктивних паралельних та розподілених систем обробки даних, застосовувати знання у практичних ситуаціях та вчитися й оволодівати сучасними знаннями.

Завдання навчальної дисципліни «Технології розподілених систем та паралельних обчислень» полягає в отриманні навичок здобувачами вищої освіти з реалізації систем розподілених об'єктів та паралельного програмування. У результаті вивчення навчальної дисципліни здобувач вищої освіти повинен знати архітектуру та програмне забезпечення високопродуктивних паралельних та розподілених обчислювальних систем, чисельних методів і алгоритмів для паралельних структур, вміти виконувати паралельні та розподілені обчислення, застосовувати чисельні методи і алгоритми для паралельних структур, мови паралельного програмування при розробці й експлуатації паралельного та розподіленого програмного забезпечення.

Практичне заняття включає проведення попереднього контролю знань, удосконалення вмінь і навичок здобувачів вищої освіти, постановку загальної проблеми викладачем та її обговорення за участю здобувачів вищої освіти, розв'язання завдань з їх обговоренням, перевіркою і оцінюванням.

Під час проведення практичних занять організовується дискусія щодо попередньо визначених тем, до яких здобувачі готують доповіді, а також обговорюються проблемні питання, на які мають бути знайдені відповіді в результаті дискусії.

З метою виявлення рівня засвоєння матеріалу викладачем проводиться перевірка і обговорення роботи, яку виконували здобувачі, а також підведення підсумків з одержанням здобувачами вищої освіти відповідної оцінки залежно від результатів виконаної роботи.

Варто зазначити, що проведення практичних занять вимагає наявності попередньо підготовленого матеріалу. За кожне практичне заняття фіксуються оцінки, що враховуються при виставленні модульної оцінки з даної навчальної дисципліни.

Самостійна робота над засвоєнням навчального матеріалу з дисципліни може виконуватися в домашніх умовах. Виконання завдань із самостійної та роботи є обов'язковим для кожного здобувача вищої освіти.

Заключною формою самостійної роботи є підготовка до поточного контролю зі змістових модулів. Вона базується на систематичному вивченні лекційного матеріалу, питань, розглянутих на заняттях, а також проблемних питань, досліджених самостійно та умінні логічно викладати їх суть.

Таким чином, практичні заняття разом з лекційним курсом і самостійною роботою формують професійну складову здобувачів вищої освіти, завдяки чому вони можуть швидше адаптуватися в умовах Європейської інтеграції.

1 ПРАКТИЧНА РОБОТА

Практичне заняття № 1
ПОНЯТТЯ АЛЬТЕРНАТИВНИХ ПОТОКІВ В ОПЕРАЦІЙНІЙ СИСТЕМІ WINDOWS, ПІДКЛЮЧЕННЯ ОСНОВНИХ БІБЛІОТЕК МОВИ ПРОГРАМУВАННЯ C++

Мета заняття – набути теоретичні знання щодо поняття консолі операційної системи Windows, розглянути її основні команди, ознайомимось з поняттям альтернативного потоку, встановити та зробити перше налаштування C++.

Після завантаження операційної системи Windows, запускаємо командний рядок з використанням комбінації клавіш Win+R. Вводимо інтерпретатор командного рядка «cmd» та натискаємо Ok. Створюємо текстовий файл з розширенням «.txt», що містить рядок «Hello, students!», а саме в командному рядку вводимо команду *copy con primer.txt* (де primer.txt – назва та розширення файлу) та натискаємо Enter.

Далі вводимо текст «Hello, students!» натискаємо Enter та комбінацію клавіш «ctrl+z» та отримуємо напис «скопійовано файлів:1». Файл primer.txt успішно створено. Далі у цьому файлі створюємо альтернативний потік з ім’ям «potok1.txt». Вводимо команду *echo This is second stream>primer.txt:potok1.txt* та тиснемо Enter. Переглянемо вміст основного і альтернативного потоків виконавши наступні команди: *type primer.txt* – виводиться рядок “Hello students!”. Якщо знову виконати команду *type primer.txt:potok1.txt* та звернутись до створеного раніше потоку з ім’ям «potok1.txt», побачимо помилку, бо команда не бачить другий потік.

Тому скористуємося командою «more» (рис. 1.1).

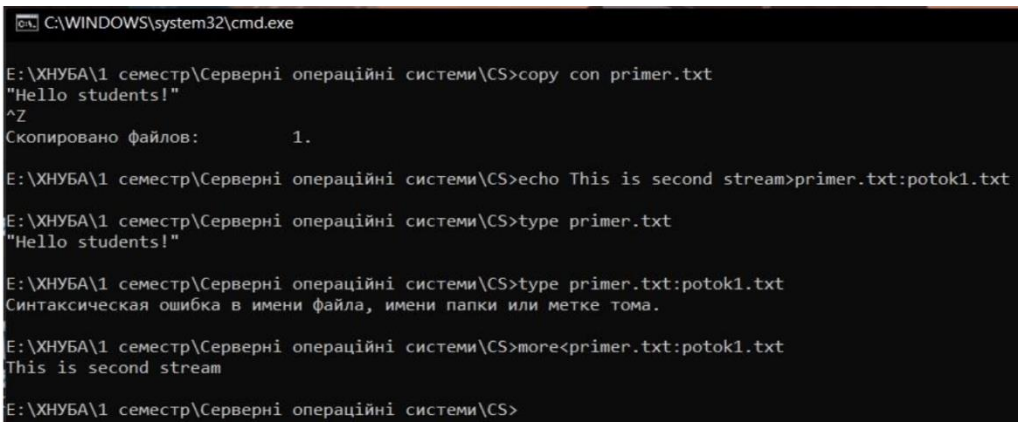

Рисунок 1.1 – Приклад створення альтернативного потоку

Створюємо в файлі другий альтернативний потік з ім’ям potok2.jpg, що містить графічний файл. Виконуємо команду *type potok2.jpg> primer.txt:potok2.jpg*. Витягуємо графічні дані з потоку за допомогою графічного редактора *mspaint primer.txt:potok2.jpg*, отримуємо графічні дані у графічному редакторі Microsoft Paint. Створюємо в файлі третій альтернативний потік з ім’ям calcul.exe, який містить програму калькулятор, а саме *type c:\Windows\system32\calc.exe>primer.txt:calcul.exe.*

Запустимо калькулятор з текстового файлу за допомогою команди *start. \primer.txt:calc.exe*. Калькулятор запустився. Послідовність дій у командному рядку описана на рисунку 1.2.

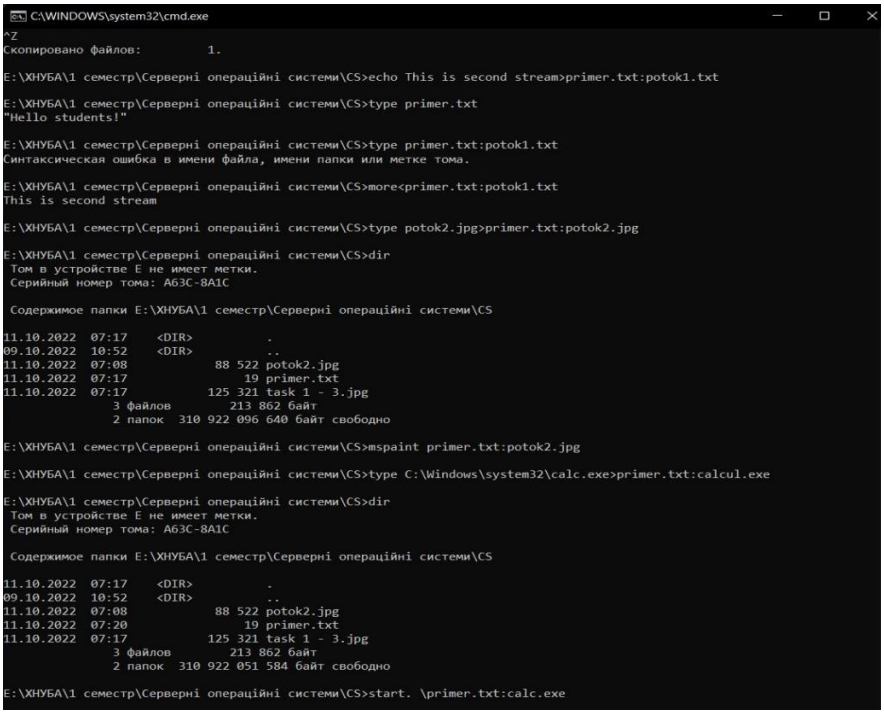

Рисунок 1.2 – Приклад створення альтернативних потоків

За допомогою інструментів Windows необхідно визначити характеристики вашого комп’ютера, а саме скільки в нього процесорів або ядер, яка їх робоча частота, скільки потоків одночасно виконується на вашому комп’ютері.

Переходимо до встановлення C++.

За допомогою посилання https://visualstudio.microsoft.com/ru/downloads/ завантажуємо Visual Studio 2022 Community та встановлюємо C++. Після інсталяції створюємо новий проєкт (рис.1.3).

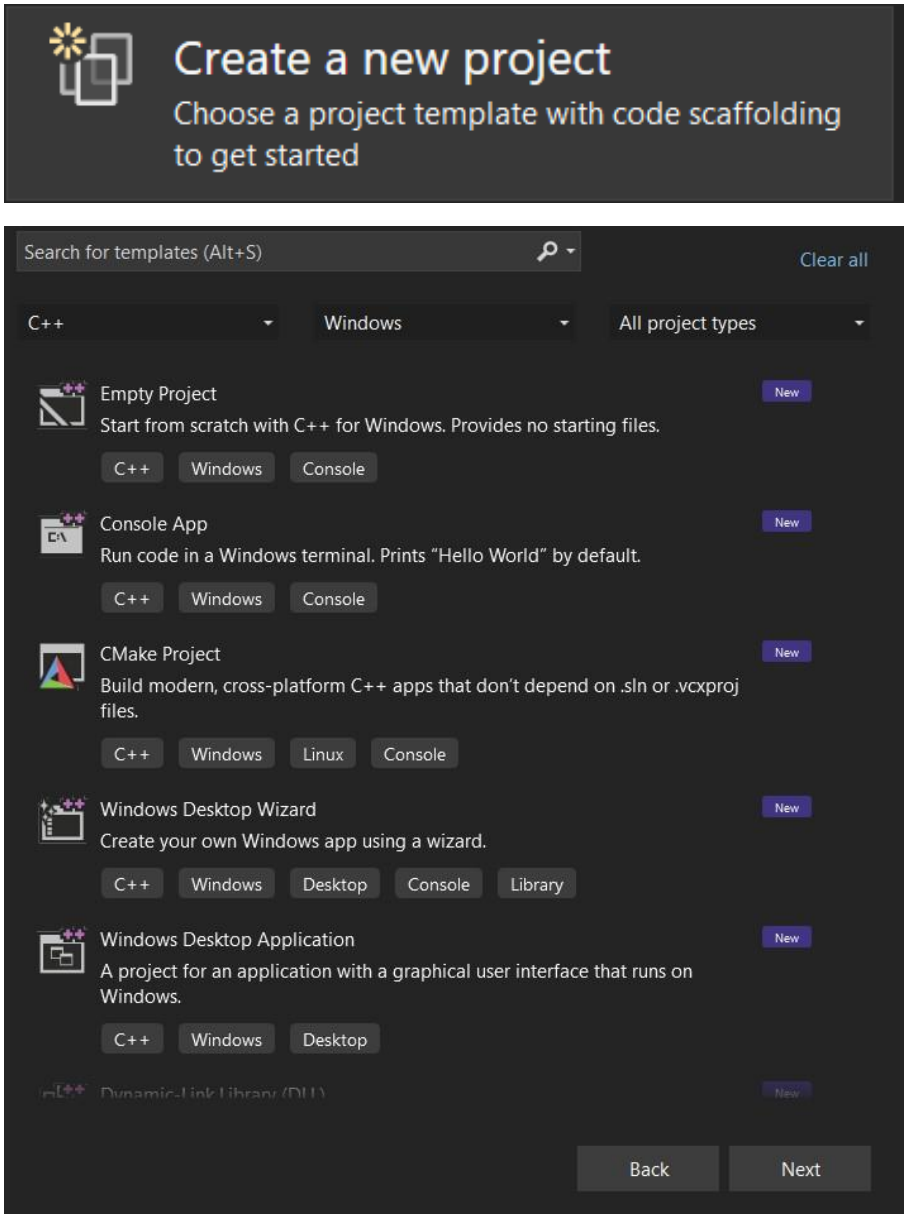

Рисунок 1.3 – Створення нового проєкту у C++

Вводимо ім’я проєкту, наприклад KN-11 lab1 (ви вводите KN-41 KNAME) (рис. 1.4).

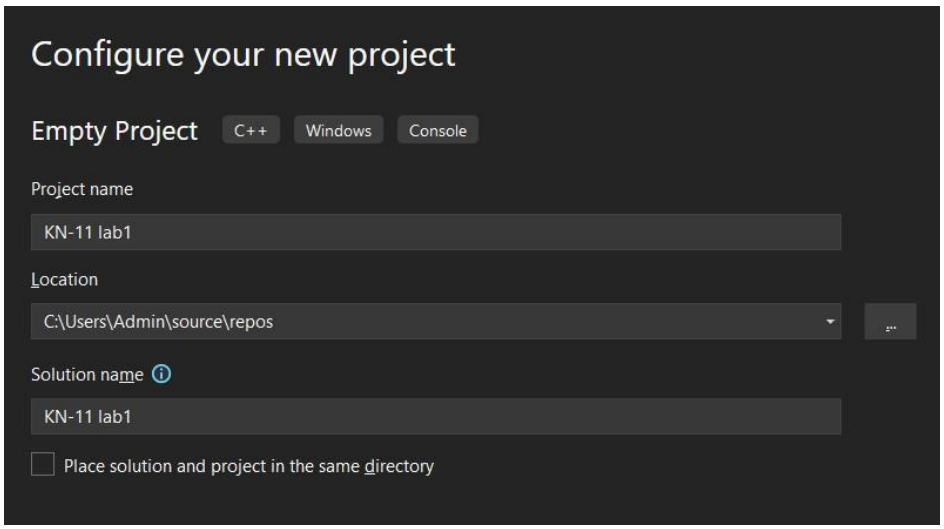

Рисунок 1.4 – Параметри створення нового файлу

Додаємо новий файл з розширенням «.cpp» (рис. 1.5).

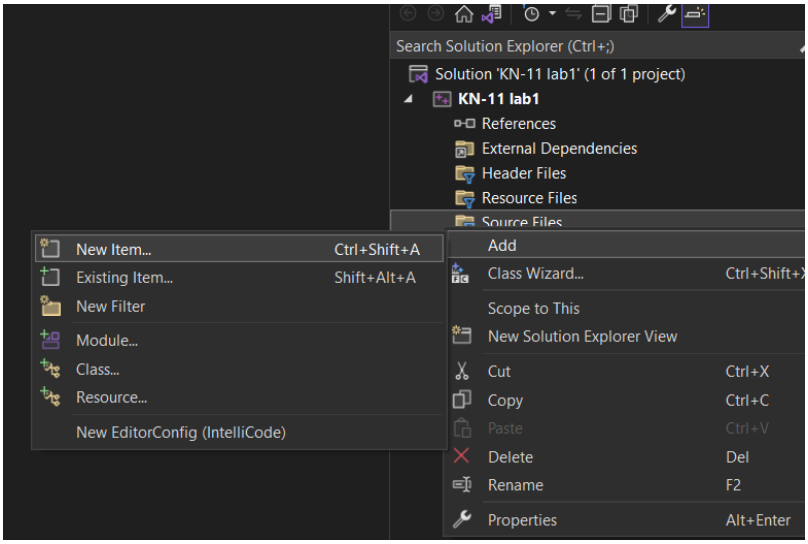

Рисунок 1.5 – Створення нового файлу з розширенням «.cpp»

Далі проводимо підключення бібліотек та виводимо тестову програму на екран:

1. Iostream – для того, щоб мати можливість використовувати метод «cout» для виводу інформації в консоль вводимо код *#include <iostream>*.

2. Кожна програма C++ повинна мати точку старту. Для цього головний (root) файл повинен містити виклик main функції: *auto main() -> int {}.*

3. Виклик «cout» для виводу тексту в консоль та «endl» для переходу у нову строку за допомогою коду *d::cout << "hello" << std::endl;*.

4. Повертаємо результат виконання функції.

5. Створюємо виконуваний файл Build-Build Solution.

6. Запускаємо програму на виконання комбінацією клавіш «CTRL+F5» і бачимо слово hello.

Запитання для самопідготовки

1. Наведіть поняття альтернативного потоку.
2. Які параметри має найсучаснішій процесор?
3. Яким чином у командному рядку вивести список існуючих команд?
4. Наведіть поняття потоку.

Практичне заняття № 2

ОСНОВИ БАГАТОПОТОКОВОГО ПРОГРАМУВАННЯ

Мета заняття – вивчити основи багатопотокового програмування на мові C++ та його використання для реалізації паралельних алгоритмів і програм.

Коли ви запускаєте будь-яку програму, операційна система створює процес, який являє собою деяку абстрактну оболонку, яка розмежовує ресурси поточної програми і ресурси інших програм, які виконуються в цій же системі.

Процес сам по собі нічого не робить, але він містить в собі потоки. Як мінімум в процесі повинен бути хоча б один потік, але їх може бути і більше. Це пов’язано з тим, що в конкретному потоці проходить виконання всієї логіки, яку ви описуєте в програмі. Потік визначає послідовність виконання коду.

За допомогою створення додаткових потоків і передачі в них частини логіки програми, наприклад, виконання деякої функції, можна забезпечити виконання коду не послідовно, а паралельно або асинхронно, таким чином прискоривши виконання програми. Аля для цього обов’язковою умовою є відповідність апаратної частини (наприклад комп’ютер з одним процесором буде перемикатися між потоками, щоб виконувати їх по черзі). Тому за цих умов ми не отримаємо приросту продуктивності.

Починаємо ввід з *#include <iostream>*. Для того щоб працювати з потоками нам необхідно підключити бібліотеку *#include<thread>* та для роботи

з часом бібліотеку *#include<chrono>*. Розглянемо деякий додатковий функціонал, який присутній у бібліотеці «thread». Нас буде цікавити можливість отримання ідентифікатора поточного потоку, щоб відрізнити один потік від іншого.

Для того щоб отримати ідентифікатор потоку, необхідно виконати метод «get_id( )», який знаходиться у просторі імен «this_thread». Для того щоб його виконати вводимо *cout << this_thread::get_id() << endl;* Наведемо приклад коду:

```
#include <iostream>
#include<thread>
#include<chrono>
using namespace std;
int main()
{
	setlocale(LC_ALL,"ru");
	cout << this_thread::get_id() << endl;
	return 0;
}
```

Коли програма запускається на виконання, то з кожним запуском ми отримуємо новий унікальний ідентифікатор потоку.

Далі з цього простору імен нам знадобиться метод «sleep_for», який призупиняє дію поточного потоку на деякий період часу. Його треба використовувати для емуляції тривалості задачі. У якості параметру він приймає час, протягом якого повинен зупинитися потік, тобто бути неактивним.

Саме на цьому етапі необхідно використовувати бібліотеку «chrono» і вказати, наприклад у мілісекундах, скільки повинен проспати потік (введемо значення 1000 мілісекунд, що дорівнює одній секунді.)

Також додамо два cout для наглядності роботи. Наведемо код до даного опису:

```
int main( )
{
setlocale(LC_ALL, "ru");
cout << "STAR MAIN" << endl;
this_thread::sleep_for(chrono::milliseconds(1000));
cout << this_thread::get_id() << endl;
cout << "END MAIN" << endl;
return 0;
}
```

Як можна побачити, спочатку виводиться надпис «START MAIN», потім йде секундна затримка, далі виводиться ідентифікатор потоку, наприкінці надпис «END MAIN». На цьому етапі програма затримує виконання коду на секунду. Таким чином, моделюється тривале виконання будь якого методу складної задачі.

Змінюємо код таким чином, щоб утворити цикл, наприклад на десять ітерацій. В кожній ітерації буде виводитися ідентифікатор потоку і ця інформація повинна бути виведена в функції «main» і призупинятися на три секунди. Наводимо код виконання:

```
int main()
{
setlocale(LC_ALL, "ru");
for (size_t i = 0; i < 10; i++)
{
cout << "ID потока = " << this_thread::get_id() << "\tmain" << endl;
this_thread::sleep_for(chrono::milliseconds(3000));
}
return 0;
}
```

Кожна ітерація, як можна побачити, зупиняється на три секунди, і програма, відповідно, виконується тривалий час. Якщо закоментувати дану

строку кодом *this_thread::sleep_for(chrono::milliseconds(3000))*, то наш код виконається миттєво.

Для того, щоб відстежити потоки, порядок їх виконання, переваги потоків, потрібно гальмувати їх виконання. Тепер змініть значення 1000 мілісекунд на 200 мілісекунд і подивіться різницю.

На наступному кроці знадобиться метод «DoWork», який буде емулювати складну задачу, яка буде виконуватися тривалий час. В цей код копіюємо наш цикл на 10 ітерацій (рис. 1.6).

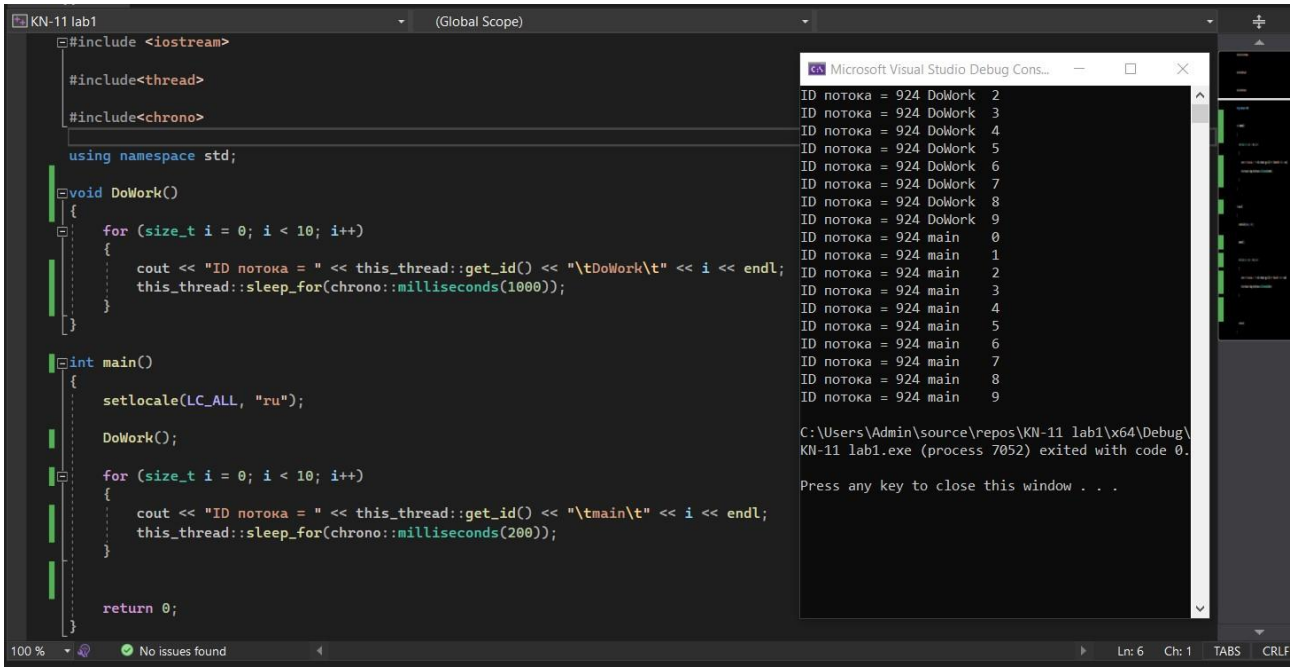

Рисунок 1.6 – Емуляція складної задачі методом «DoWork»

Як ми бачимо, спочатку виконується метод «DoWork», потім виконується метод «main», додавши нумерацію ітерацій. Зверніть особливу увагу на ідентифікатор потоку, значення при виконанні методів не змінюється. Це значить що код у методі «DoWork» та у методі «main» виконувався в одному потоці. Змінимо час виконання у DoWork на 3000 мілісекунди, а у main на 100 мілісекунд. Зверніть увагу на час виконання.

Таким чином, методом «DoWork» змодельовані деякі навантаження, наприклад складні розрахунки, запити бази даних та обробка цієї інформації, або запити до віддалених серверів.

Зауважимо, що в таких ситуаціях можна використовувати додаткові потоки, щоб розпаралелити ці дві задачі. Можна примусити задачу виконувати метод «DoWork» окремо. Таким чином, якщо в нас буде достатня кількість апаратних ресурсів, то ми отримаємо прискорення виконання програми, що виконується.

Створимо екземпляр потоку «thread», причому передаємо до конструктора цього потоку вказівник на функцію, яка буде виконуватися паралельно. В нашому випадку можна передати просто ім’я функції «DoWork» без круглих дужок. На цьому етапі не викликаємо функцію, а просто передаємо вказівник на функцію і ім’я функції це також вказівник на функцію *thread th(DoWork);*. У момент створення об’єкту «th» і передачі йому, в якості вказівника, функції «DoWork», створюється окремий потік і цей метод вже буде виконуватися в окремому потоці, але цього не достатньо для повнофункціональної роботи програми, особливо плутаниця методів.

Коли ми створюємо об’єкт «th», при цьому створюється новий системний потік «DoWork», який зв’язаний з об’єктом «th». Щоб уникнути помилки, яка виникла на попередньому кроці виконання програми, необхідно визначитися, що нам робити з цим потоком і об’єктом (рис. 1.7).

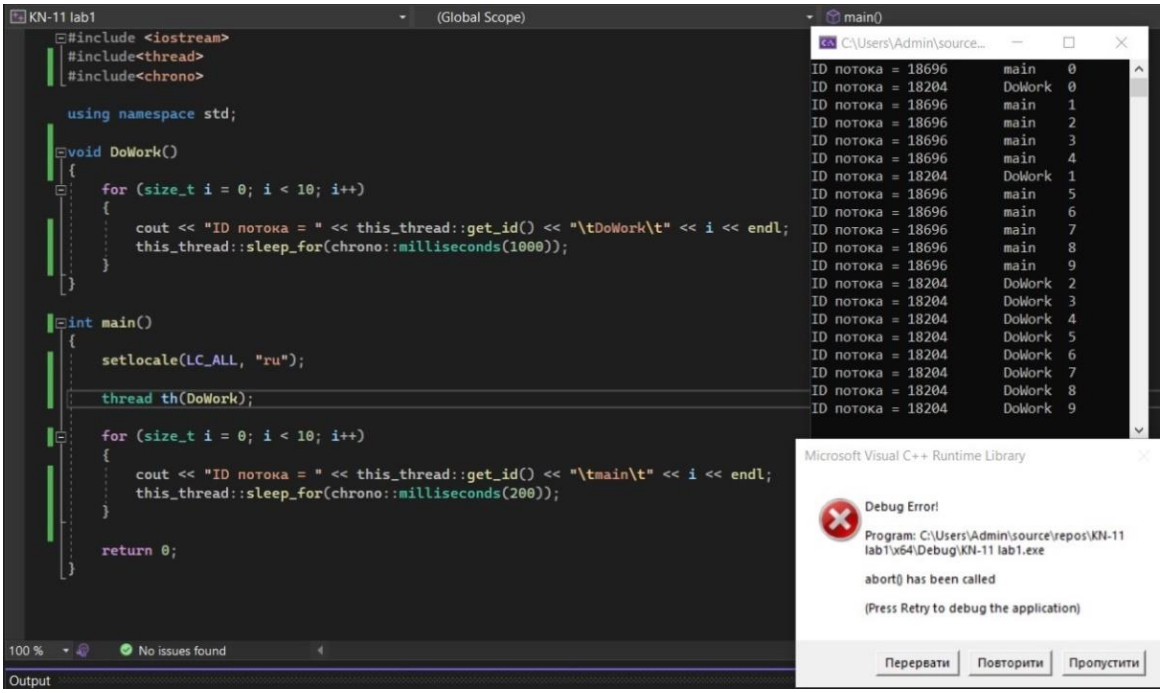

Рисунок 1.7 – Приклад некоректної роботи програми

Для цього існують методи «detach» і метод «join». Detach допомагає розірвати зв’язок між об’єктом «th» і тим потоком, який ми почали виконувати. Недоліком даного методу є закінчення роботи програми при закінченні виконання одного з потоків. У зв’язку з тим, що ми маємо різний час затримки потоків, то при повному виконанні методу «main», DoWork майже не буде виконаний.

Нижче застосовуємо метод «join( )» (рис. 1.8).

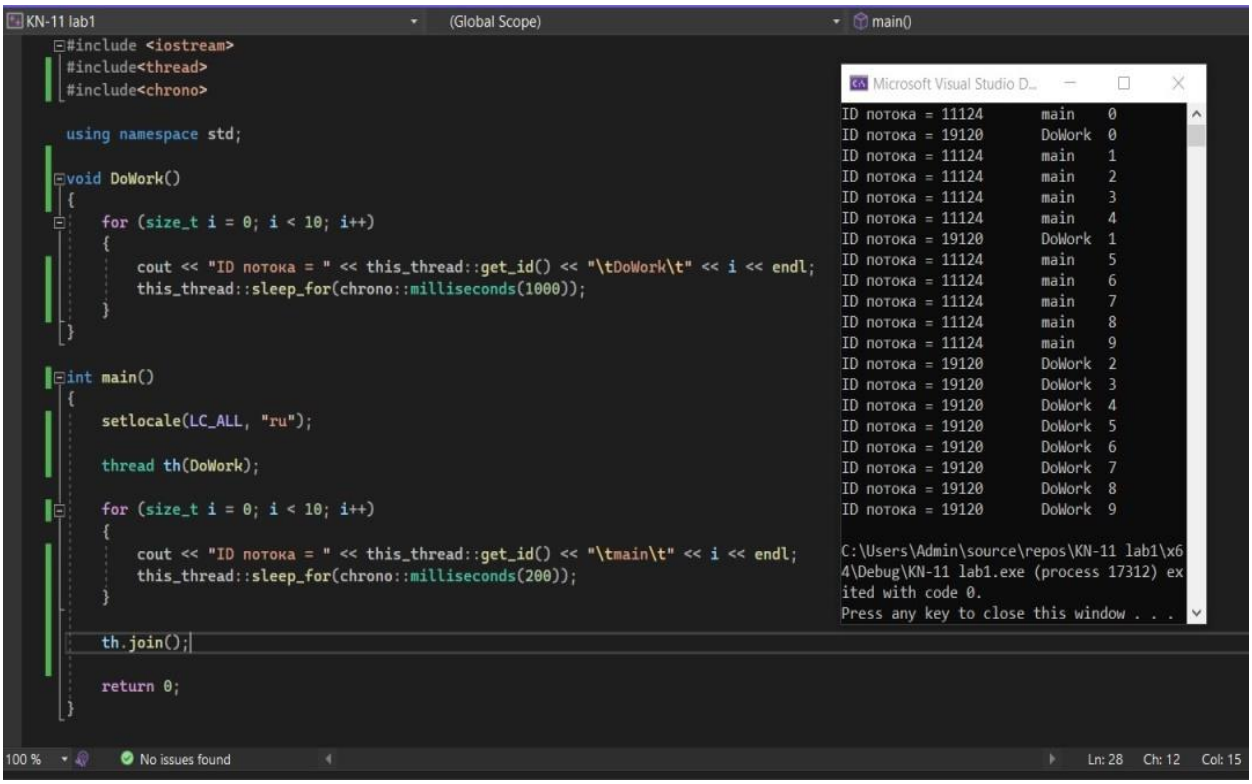

Рисунок 1.8 – Застосування методу «join( )»

У зв’язку з тим, що виконання main виконується у п’ять разів швидше ніж DoWork (значення 200 порівняно з значенням 1000), на рисунку 1.8 можна побачити повне коректне виконання обох потоків.

Таким чином, коли ми створюємо потік, необхідно задати параметри його поведінки методом «detach» або методом «join». У підсумки ми отримуємо виконання двох методів паралельно.

Створимо третій потік, назвемо його «th2» і передамо йому метод «DoWork» (рис. 1.9).

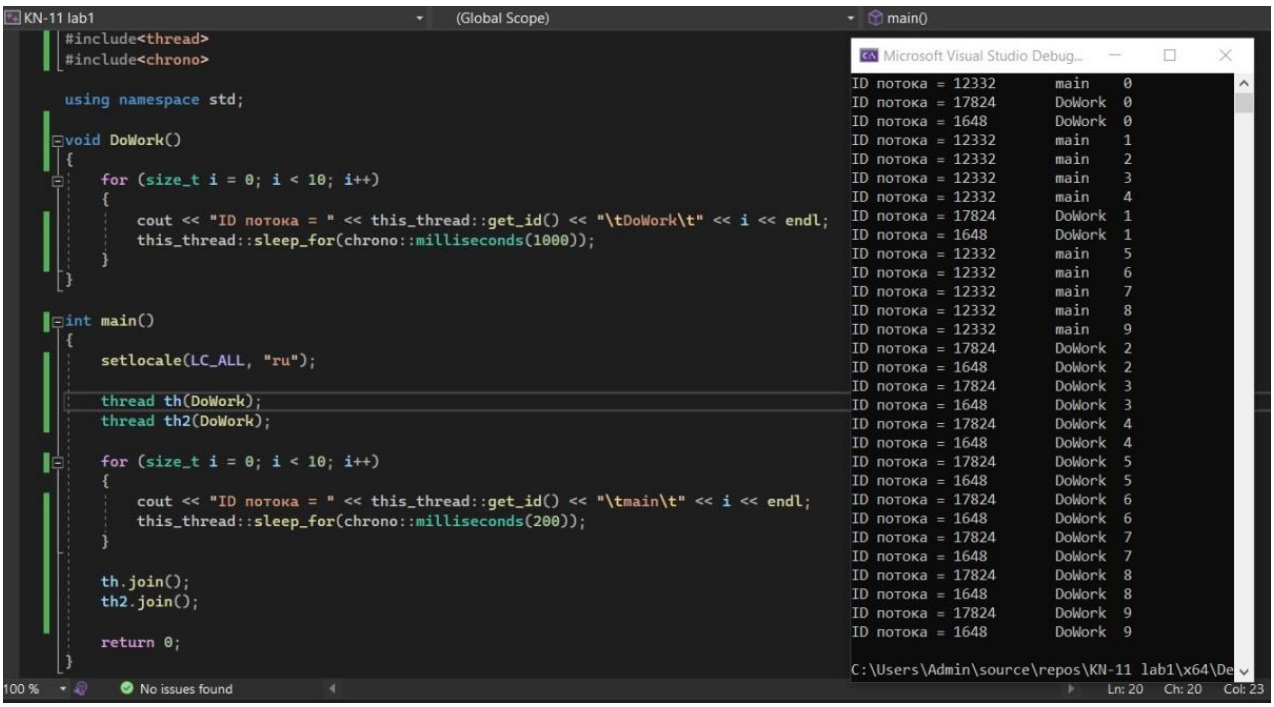

Рисунок 1.9 – Приклад створення третього потоку

У підсумку виконується метод «main» і два методи «DoWork», причому у кожного методу «DoWork» свій власний ідентифікатор. Таким чином можна запускати у декількох потоках стільки задач, скільки нам необхідно.

Наприкінці необхідно до кожної строки коду програми зробити пояснення відповідної функції, бібліотеки, методу тощо.

Також необхідно зробити покроковий звіт з описом послідовності дій і відповіддю на контрольні питання.

Запитання для самопідготовки

1. Для чого використовується багатопотокове програмування?
2. Наведіть основні відмінності методів «detach» і « join».
3. Який принцип створення циклів у C++?
4. Для чого у коді програми використовується бібліотека «chrono»?
5. Який метод призупиняє дію потоку на деякий період часу?
6. Надайте визначення поняття «потік».

Практичне заняття № 3

БАГАТОПОТОКОВЕ ПРОГРАМУВАННЯ: ПЕРЕДАЧА ПАРАМЕТРІВ У ПОТІК

Мета заняття – опанувати передачу параметрів у потік за допомогою C++.

Розглянемо елементарну задачу суми двох чисел. Наприклад, нам необхідно виконати цю задачу в окремому потоці, емулюючи навантаження на цю задачу тим, що будемо призупиняти роботу потоку на деякий проміжок часу. Але для цієї задачі знадобиться передавати вихідні дані, тобто числа, суму яких ми будемо знаходити. Ці числа будемо додавати у функцію *DoWork: void DoWork (int a, int b)*. У самій функції при цьому, будемо використовувати затримку виконання на одну секунду за допомогою коду *this_thread::sleep_for (chrono::milliseconds(1000)*);.

Далі створимо службові повідомлення, які не потрібні для роботи функції, але потрібні користувачеві, щоб отримувати інформацію у якому порядку проходить виконання. Тому створимо запис, який буде повідомляти у консоль про те, що почалося виконання функції *DoWork: cout << "****\t" << "DoWork STARTED\t****" << endl;*. Далі призупиняємо потік ще на дві секунди кодом *this_thread::sleep_for(chrono::milliseconds(2000)*. Тобто ми емулюємо ситуацію, що наша функція складно стартувала і проходить виконання складних дій. В нашому випадку «складними діями» буде знаходження суми.

Далі виводимо на екран виконання операції *cout << "a+b=" << a + b << endl;*. Після цього ще раз призупиняємо потік на одну секунду *this_thread::sleep_for(chrono::milliseconds(1000))* та виводимо повідомлення про те, що наш метод, який виконувався в цьому потоці завершений *cout << "****\t" << "DoWork STOPPED\t****" << endl;*. Т

Таким чином ми емулюємо складну роботу розрахунків протягом чотирьох секунд (рис. 1.10).

```
void DoWork(int a, int  b)

{

    this_thread::sleep_for(chrono::milliseconds(1000));

    cout << "****\t" << "DoWork STARTED\t****" << endl;

    this_thread::sleep_for(chrono::milliseconds(2000));

    cout << "a+b=" << a + b << endl;

    this_thread::sleep_for(chrono::milliseconds(1000));

    cout << "****\t" << "DoWork STOPPED\t****" << endl;

}
```

Рисунок 1.10 – Емуляція складних розрахунків протягом 4-х секунд

Якщо виконувати DoWork у main, то наш потік блокувався би і ми нічого не зможемо робити з нашою програмою протягом 4-х секунд. У методі «main» задача емулювати постійну роботу (може відбуватись обслуговування запитів користувачів, робота з консоллю тощо). На даному етапі задача показати неперервну роботу в основному потоці (рис. 1.11).

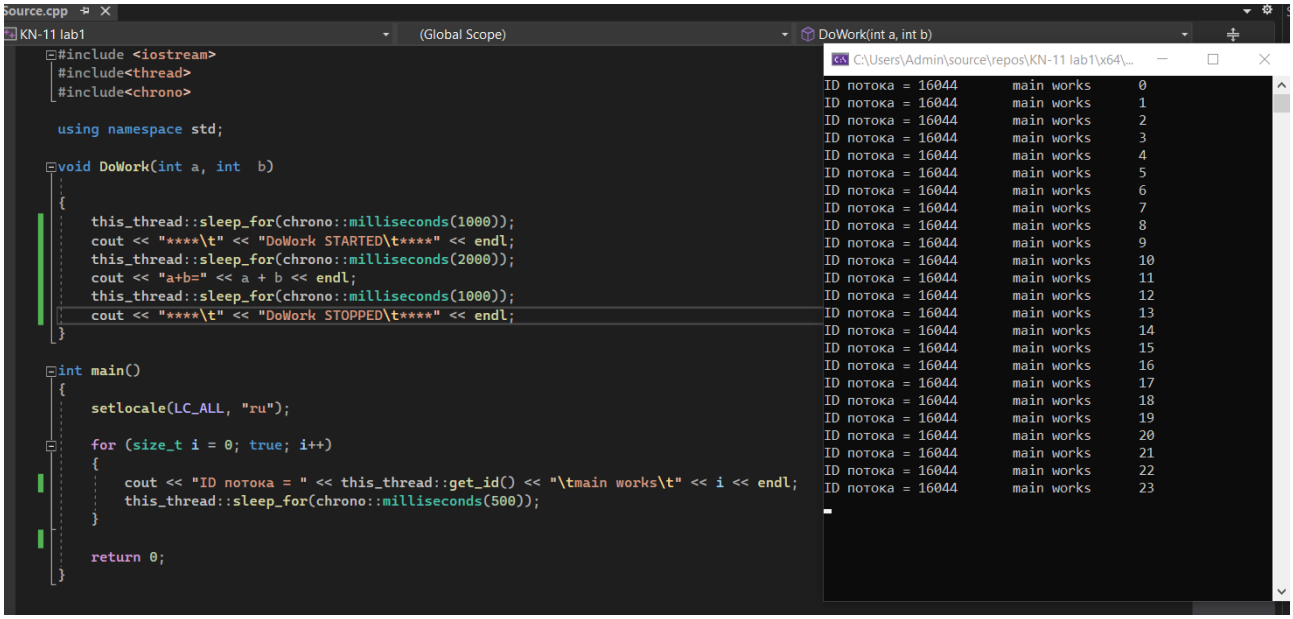

Рисунок 1.11 – Приклад неперервної роботи в основному потоці

Тепер звернемося до методу «DoWork» (складна математична операція на чотири секунди). Зробимо обчислення 5 плюс 5 (рис. 1.12), причому зверніть увагу на час виконання обчислень.

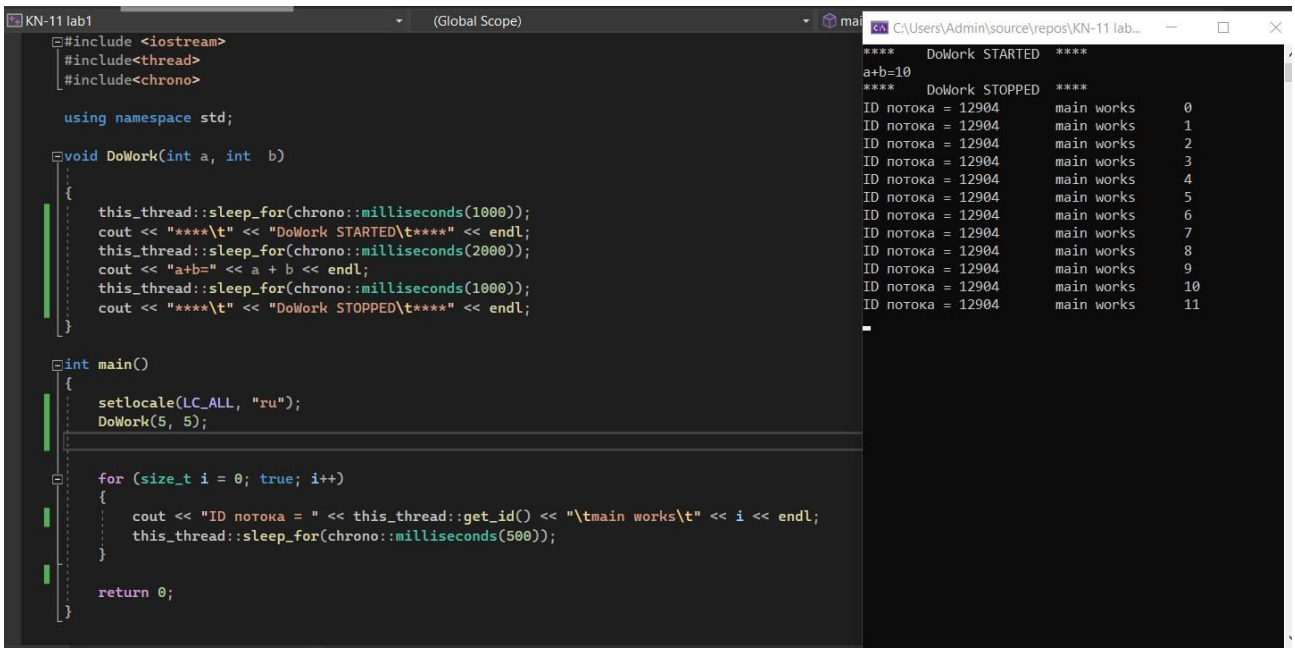

Рисунок 1.12 – Виконання складних розрахунків

Незручно очікувати стільки часу на виконання операції. Для цього математичну операцію виконуємо в окремому потоці. Потоку у конструктор передаємо назву функції, яку ми бажаємо виконувати у потоці, не забуваємо використати *th.join();*, хоча при нашій умові циклу *for (size_t i = 0; true; i++)* не можна попасти в рядок коду join, оскільки цикл в нас нескінченний, але і без join або detach потік неможна використовувати.

За звичайних правил ви повинні були б в функцію, яку викликаємо у круглих дужках передати параметри, але не у випадку з потоком. Це пов’язано з тим, що потік приймає не функцію, що викликається, а сам вказівник на функцію і вже сам в окремому потоці її викликає. Тому для потоку в конструкторі передбачено наступний функціонал. Спочатку передається назва тієї функції, яка виконується, а далі підряд параметри, які попадуть в цю функцію.

В тестовому прикладі це функція з двома параметрами, томи передаємо довільні значення для нашої математичної операції *thread th(DoWork, 5, 5);*. Таким чином потік знає, що повинен викликати метод «DoWork» у перший параметр а він передасть число 5 і у другий параметр він передасть число 5 (рис. 1.13).

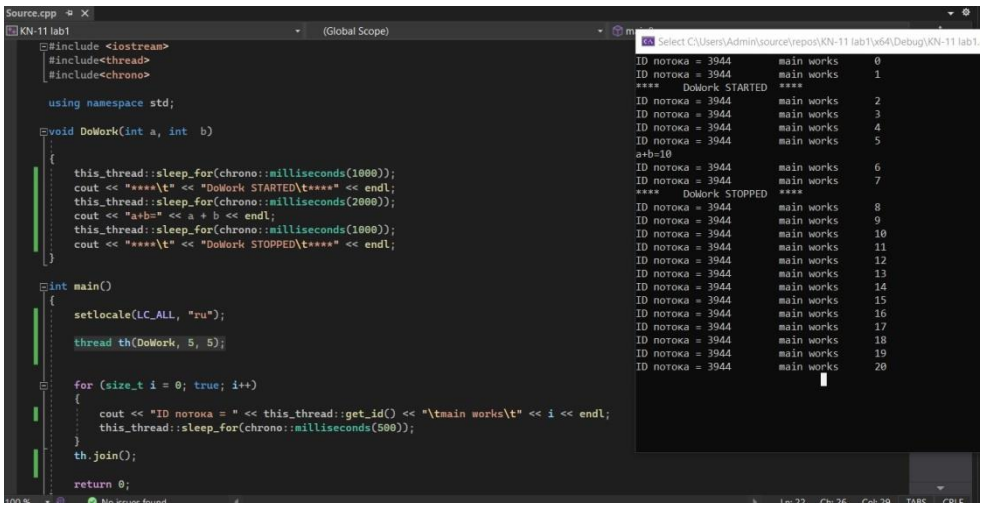

Рисунок 1.13 – Підсумки розрахунків

Як можна побачити, до того як стартував DoWork, вже було виконано дві ітерації нескінченого циклу. Найголовніше те, що нашими розрахунками ми не зупиняли роботу основного потоку «main».

Зауважимо, що кількість параметрів контролюється середою розробки та компілятором і воно завжди повинно відповідати типу даних, які приймає у відповідних параметрах функція, яку ви викликаєте в потоці, а також кількості цих параметрів.

Додамо третій параметр, а саме повідомлення *void DoWork(int a, int b, string msg)*. Тепер функція «DoWork» приймає три параметри *thread th(DoWork, 5, 5, "massage KN-41 KNAME");*. Для пріоритету виводу текстового повідомлення вводимо строку *cout << msg << endl;*. Зовнішній вигляд роботи функції з трьома параметрами наведено на рисунку 1.14.

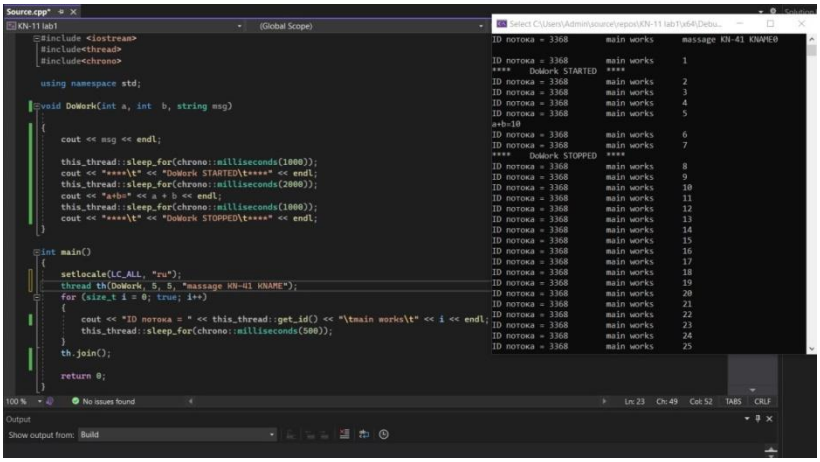

Рисунок 1.14 – Зовнішній вигляд роботи функції з трьома параметрами

Саме таким чином організована передача параметрів в потік.

Наприкінці необхідно до кожної строки коду програми зробити пояснення відповідної функції, бібліотеки, методу тощо. Також необхідно зробити покроковий звіт з описом послідовності дій і відповіддю на запитання до самопідготовки.

Запитання для самопідготовки

1. Яку кількість змінних можна додати до функції замість a та b в рядку коду програми *DoWork: void DoWork (int a, int b)*?

2. Чому повинна відповідати кількість параметрів, наведених в питанні 1?

3. Назвіть відомі вам бібліотеки C++.

4. Наведіть свій варіант складних дій та опишіть результат його виконання.

5. Що називають паралельними алгоритмами?

Практичне заняття № 4

ПОВЕРНЕННЯ РЕЗУЛЬТАТУ З ПОТОКУ ЗА ПОСИЛАННЯМ

Мета заняття – опанувати передачу параметрів у потік за допомогою програми C++.

У третьому практичному занятті було розглянуто, яким чином можна передати параметри для функції, якщо цю функцію необхідно виконувати в окремому потоці. Але необхідно отримати результат роботи функції з потоку.

Це можна зробити, наприклад, передавши параметри в поток за посиланням. Виконаємо додавання двох чисел в окремому потоці (рис. 1.15).

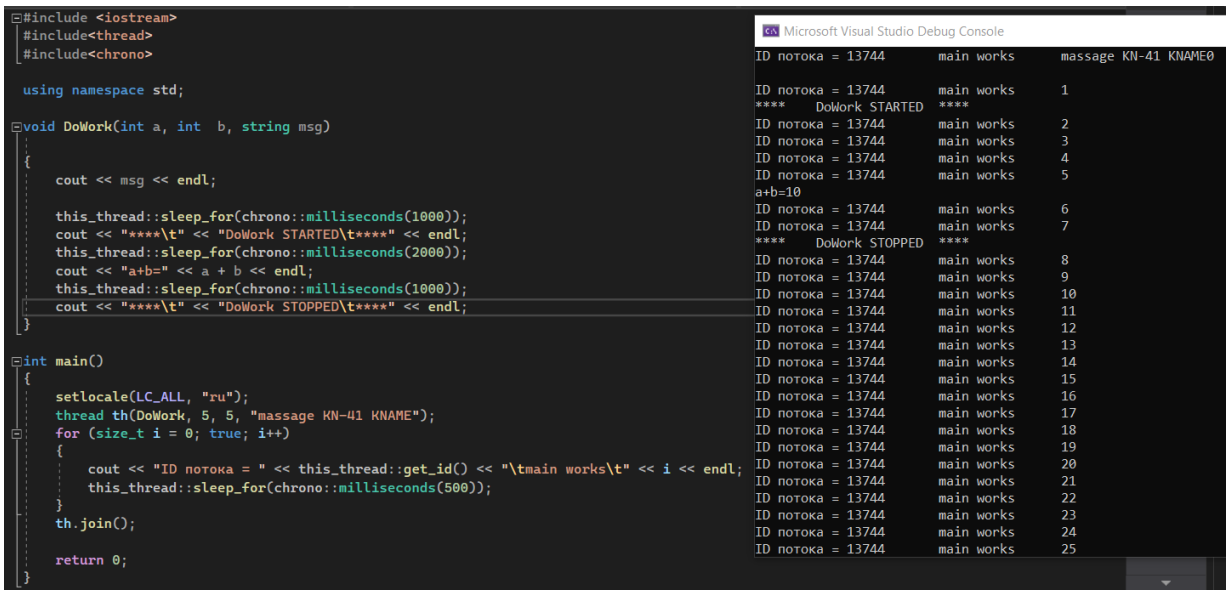

Рисунок 1.15 – Результат додавання двох чисел

Мови програмування вчать нас, що одна функція, яка використовується у коді, повинна виконувати лише одну задачу. Як можна побачити, *void DoWork (int a, int b)* виконує дві задачі: робить підрахунок суми та виводить інформацію на консоль. Для конкретного розробника програмного продукту, у даному випадку, може не знадобиться вивід на консоль результатів розрахунку.

Для коректного алгоритму підрахунку необхідно, щоб суму двох чисел рахувала функція «Sum», і окрема функція для виводу у консоль. Посилання чи вказівник використовуються, коли в одну зі змінних, в яку передаються параметри, повернути результат її підрахунку.

Розглянемо передачу параметрів в функцію за посиланням. Наприклад, будь-яке число, яке буде передаватися в нашу функцію «DoWork», буде помножуватися на 2. Ми отримуємо значення за посиланням, помножуємо його на два і у ту саму змінну його привласнюємо. Оскільки число ми отримуємо за посиланням, то всі зміни, що відбуваються з цим числом, відобразяться на вхідному параметрі. Також створимо змінну, наприклад, q та привласнимо їй значення 5 і викликаємо функцію «DoWork» (поки без потоків) і передамо їй змінну q.

Після завершення DoWork виведемо значення q на консоль (рис. 1.16).

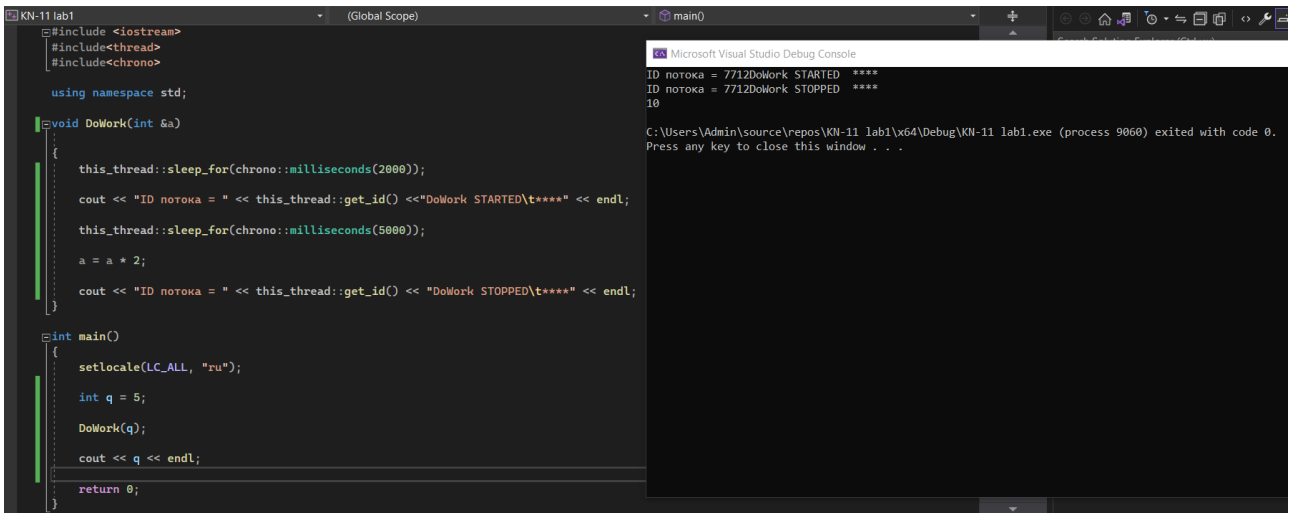

Рисунок 1.16 – Створення стартового програмного коду

У підсумку бачимо, що DoWork імітує важку роботу. Значення q у підсумку стало дорівнювати десяти, бо ця змінна подвоїлася. Оскільки імітоване складне виконання, то операція подвоєння зайняла сім секунд.

Далі додаємо цикл і запускаємо програму на виконання (рис. 1.17).

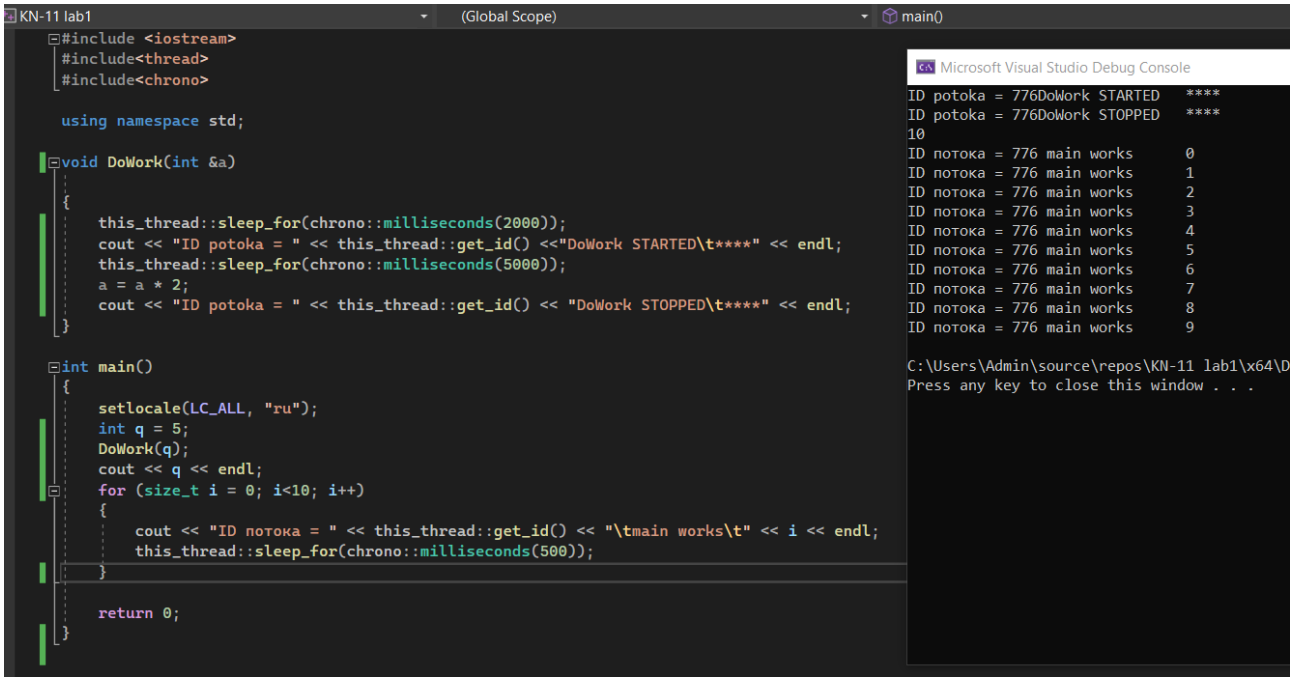

Рисунок 1.17 – Додавання циклу до програмного коду

Як можна побачити, виконання циклу блокується, поки не буде виконаний DoWork. Має смисл використовувати окремий потом для DoWork. Але виникає питання, як передати за посиланням значення в наш потік і як отримати значення нашої роботи? Нам необхідні вихідні дані, які ми будемо передавати параметрам в функцію.

Зауважимо, що у випадку, коли ми використовуємо просто функцію «DoWork» без потоку, ми можемо передати їй параметром змінну. У зв'язку з тим, що ми її передаємо за посиланням, то у підсумку роботи функції «DoWork» ми отримаємо результат в нашу зміну q назад, тобто всі зміни, які відбудуться з даним параметром в DoWork відобразиться на змінній q.

Але у випадку з потоком це працює не так. Якщо у якості другого параметра вказати змінну q (хоча DoWork приймає параметр за посиланням) і це все виконується у потоці, то за замовчуванням він передає параметр за значенням. Додамо виконання методу «join» підсумуємо (рис. 1.18).

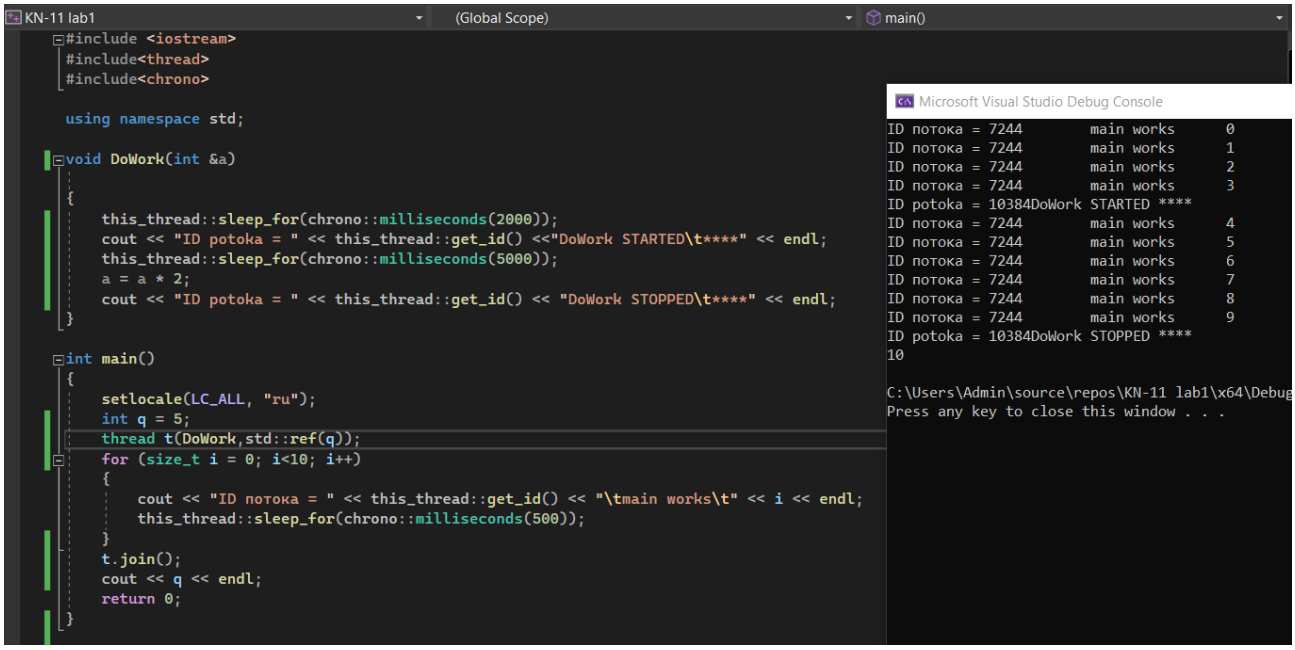

Рисунок 1.18 – Додавання методу «join»

Як можна побачити, виконується main потік, потім стартує DoWork, потім відпрацьовує main потік у кількості десяти ітерацій, але є метод «join», який очікував виконання основного потоку. І вже після того, як він виконався, DoWork зупинився і у підсумки отримали значення десяти через посилання.

Зазначимо, що задача функції «ref» полягає у створенні класу «reference_wrapper», який дозволяє працювати з потоком таким чином, щоб отримати з нього результат його роботи через посилання.

Таким чином, якщо необхідно повернути результат роботи потоку, методу або функції, який виконується в потоці і яка приймає параметр за

посиланням, необхідно використовувати ref для змінної, яка передається в якості параметра в ту функцію, яка працює в окремому потоці.

Використаємо метод «detach» і запустимо програму на виконання (рис. 1.19).

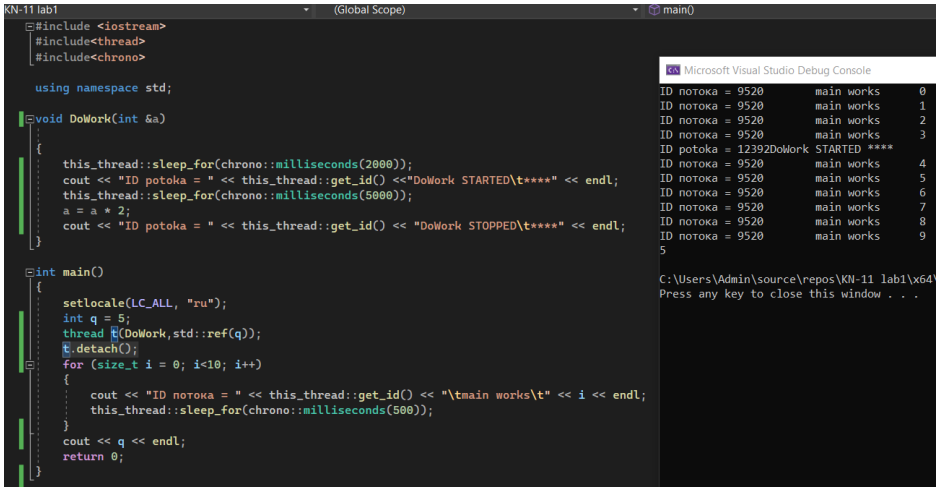

Рисунок 1.19 – Додавання методу «detach»

Як можна побачити, результат роботи дорівнює п’яти замість десяти, бо програма не дочекалась моменту виконання методу, а просто вивела q=5.

Запитання для самопідготовки

1. Що вам відомо про допоміжну функцію «ref»?
2. Наведіть визначення потоку.
3. У чому полягає різниця між методом та функцією?
4. Які значення можуть виступати у якості параметрів в С++?

Практичне заняття № 5

ЛЯМБДА-ВИРАЗ ТА ПОВЕРНЕННЯ РЕЗУЛЬТАТІВ ВИКОНАННЯ ПОТОКІВ

Мета заняття – опанувати поняття «анонімні функції» засобами С++.

На минулому практичному занятті було розглянуто отримання результатів роботи функції, яка виконувалась в окремому потоці і не блокувала основний потік. Результат робото функції було організовано за допомогою передачі параметрів за посиланням.

Для роботи по тематиці даного практичного заняття необхідно використовувати функцію «Sum», яка буде емулювати виконання складної задачі за допомогою методу «sleep_for».

Задамо виконання знаходження суми двох чисел протягом семи секунд (рис. 1.20).

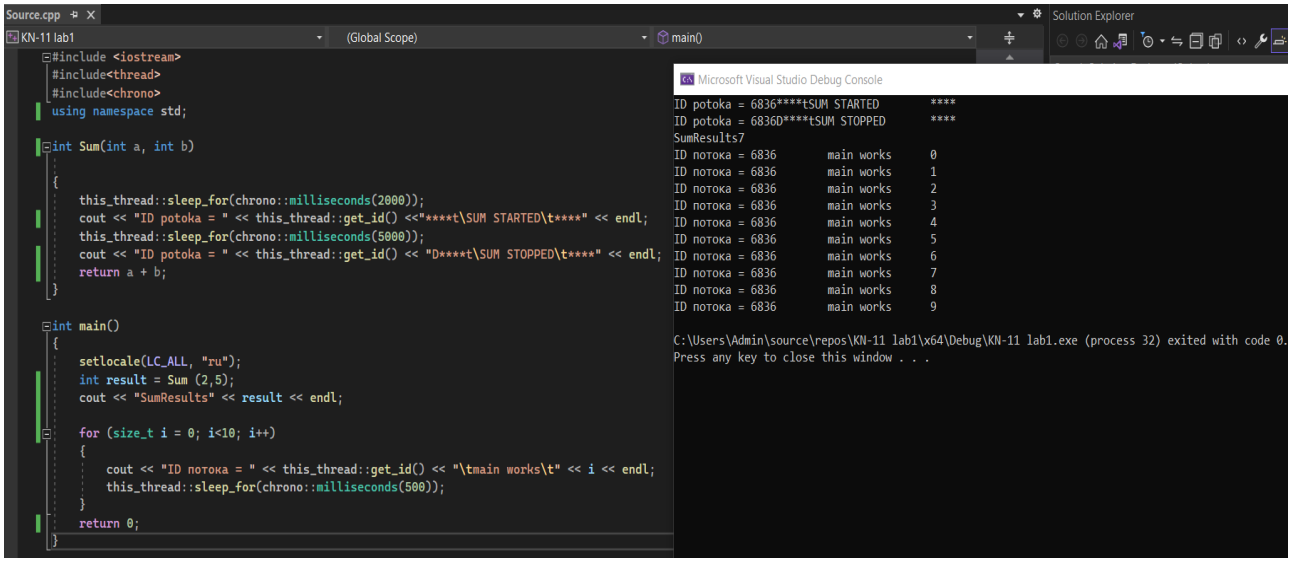

Рисунок 1.20 – Реалізація використання функції «Sum»

Після запуску програми на виконання, можна побачити, що потік 7 секунд нічого не робить.

Запустимо функцію «Sum» в окремому потоці, щоб задача, яка тривала більше, виконувалась в окремому потоці. А ще краще запускати декілька функцій в різних потоках (за умови апаратної відповідності).

Створимо потік «t», в який передамо функцію «Sum» та параметри (наприклад 2 та 5).

Після того, як відпрацює основний код, необхідно дочекатися завершення виконання результатів розрахунку (метод «join»).

Постає питання. Яким чином з функції «Sum» , яка повертає int, після виконання в потоці, отримати результати?

У коді, наведеному на рисунку 1.21 ніяк, бо потік, не зважаючи на те, що виконує функцію «Sum», не блокує основний потік, а отже і результат не буде отримано.

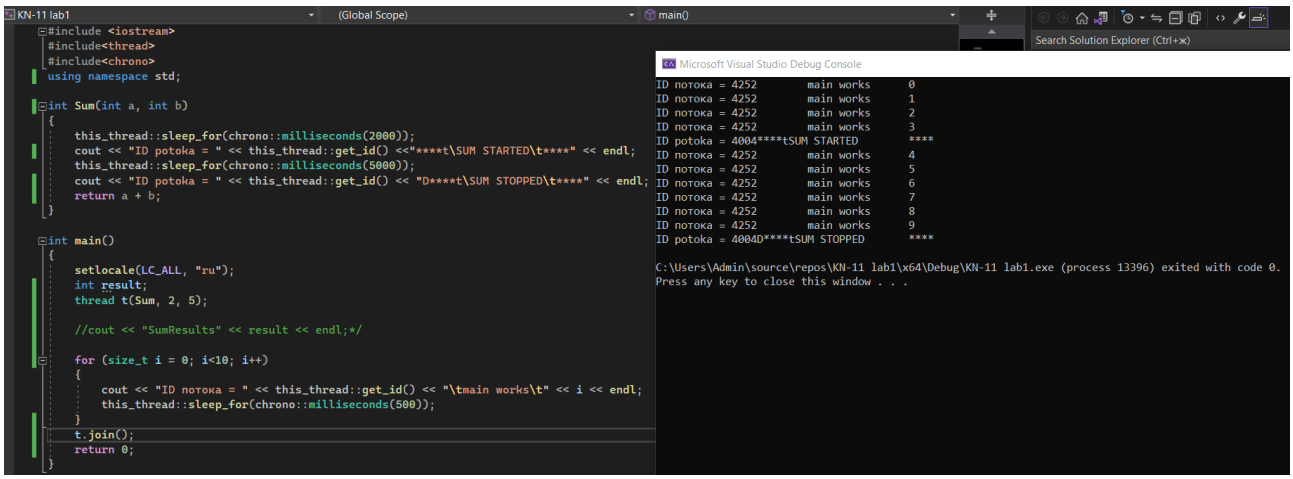

Рисунок 1.21 – Створення потоку «t», в який передано в функцію «Sum» та параметри 2 та 5

Для вирішення цієї проблеми необхідно використати анонімну функцію (лямбда-функція, лямбда-вираз). Для того, щоб отримати у змінну «result» результат виконання в окремому потоці функції «Sum», використаємо лямбда-вираз (рис. 1.22).

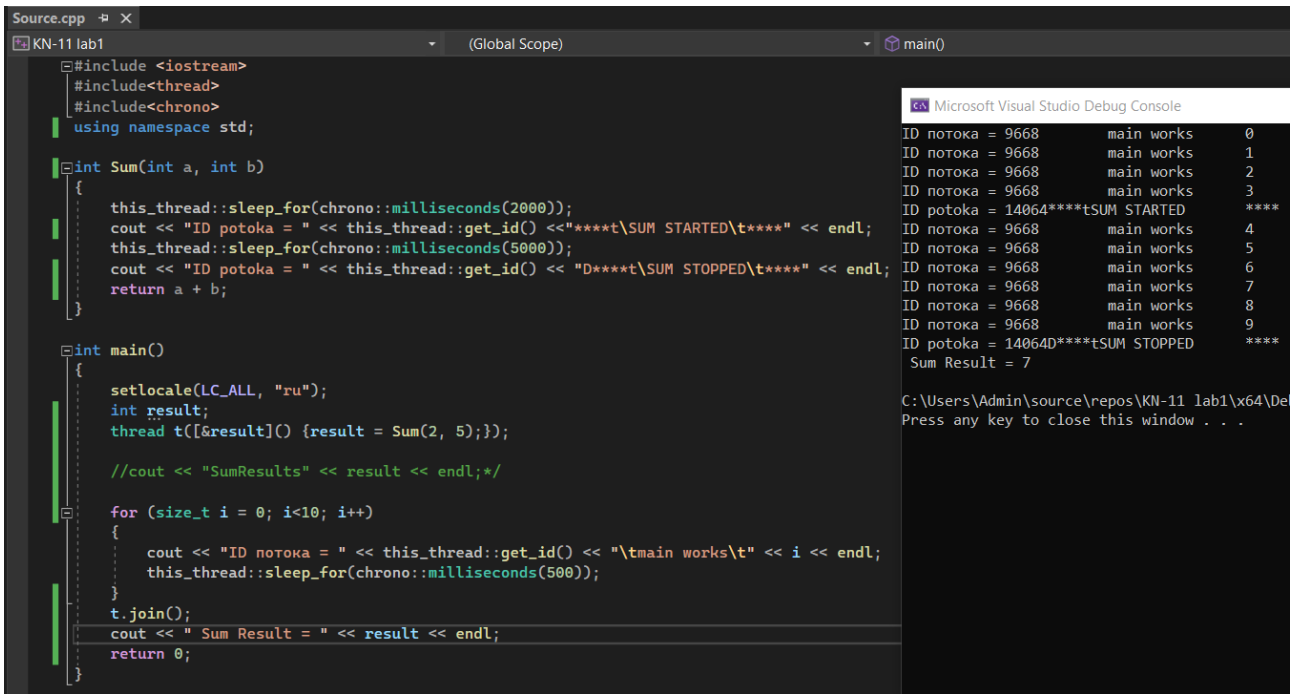

Рисунок 1.22 – Використання анонімної функції

Завдяки даному алгоритму можна запускати будь-яку функцію, яка повертає значення в окремому потоці і отримати результат її роботи. В ході виконання роботи, у потік було передано лямбда-вираз.

Також можна даний вираз записати в іншому вигляді (рис. 1.23).

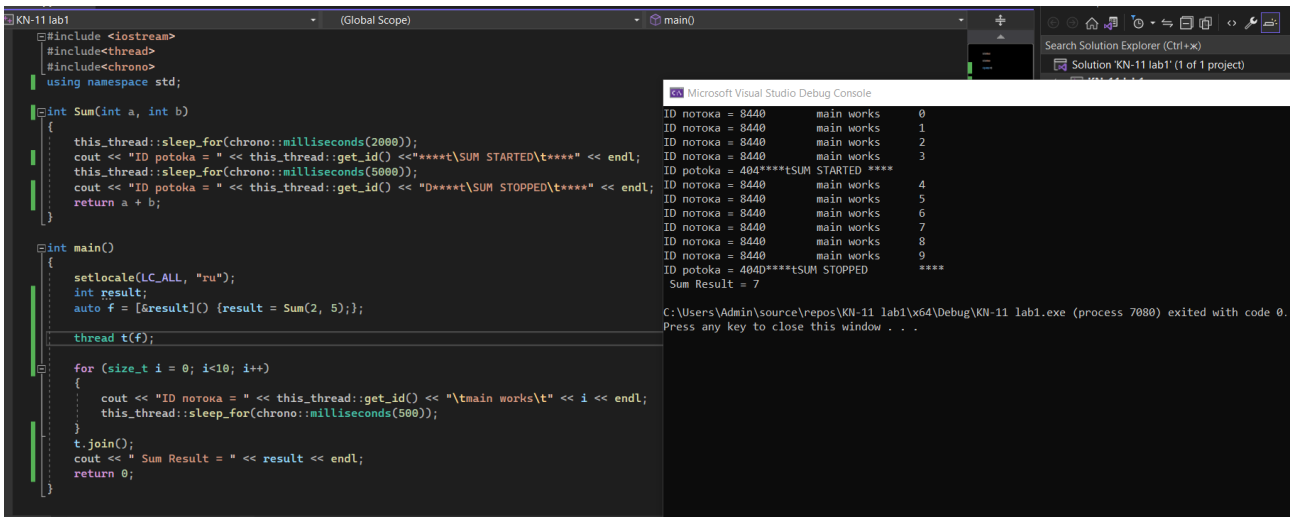

Рисунок 1.23 – Використання анонімної функції та «auto function»

Змінивши код, можна досягти результату, який наведено на рисунку 1.24.

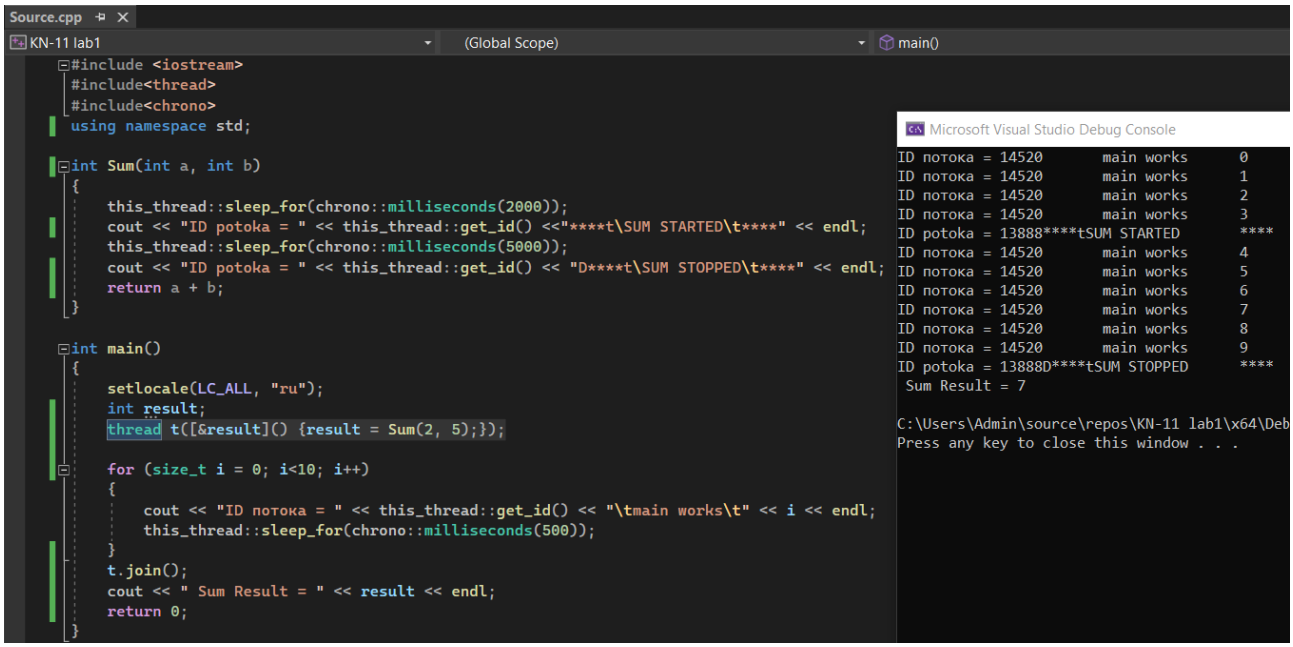

Рисунок 1.24 – Альтернативне використання анонімної функції та методу «auto function»

Запитання для самопідготовки

1. Наведіть синтаксис лямбда-виразу.
2. Опишіть призначення auto function у C++ та сферу використання.
3. Наведіть практичну користь від використання лямбда-виразів.
4. Для чого існує рядок: *this_thread::sleep_for(chrono::milliseconds(500));*?

Практичне заняття № 6

ПОТОКИ ТА МЕТОДИ КЛАСУ

Мета заняття – набути навичок використання об’єктів класу, передачі методів класу в окремий потік.

Раніше було розглянуто роботу в потоці з функцією. C++ це об’єктно-орієнтована мова програмування, тому виникають ситуації, коли необхідно передати в потік об’єкти класів. Об’єкти класів, в свою чергу, є методами цього класу і при роботі з потоками є деякі синтаксичні відмінності.

Створюємо клас «MyClass», в якому є метод «DoWork», який не приймає жодних параметрів і не повертає жодних результатів. Іншими словами, моделюємо довготривале завдання, яке не буде блокувати основний потік. Також в класі «MyClass» є метод «DoWork2», який є аналогом методу «DoWork», але він приймає один параметр, виконуючи довготривале завдання і буде виводитися значення параметру в рамках цього методу. Третій метод в класі «MyClass» це метод «Sum», завдання якого полягає в додаванні двох чисел. В main знаходиться цикл на десять ітерацій, починаючи з одиниці (рис. 1.25).

Source.cpp

KN-11 lab1 | (Global Scope) | main()

```
class MyClass
{
public:
    void DoWork()
    {
        this_thread::sleep_for(chrono::milliseconds(2000));
        cout << "ID potoka = " << this_thread::get_id() << "****t\DoWork STARTED\t****" << endl;
        this_thread::sleep_for(chrono::milliseconds(5000));
        cout << "ID potoka = " << this_thread::get_id() << "****t\DoWork STOPPED\t****" << endl;
    }
    void DoWork2(int a)
    {
        this_thread::sleep_for(chrono::milliseconds(2000));
        cout << "ID potoka = " << this_thread::get_id() << "****t\DoWork2 STARTED\t****" << endl;
        this_thread::sleep_for(chrono::milliseconds(5000));
        cout << " DoWork2 значення параметра\t " << a << endl;
        cout << "ID potoka = " << this_thread::get_id() << "****t\DoWork2 STOPPED\t****" << endl;
    }
    int Sum(int a, int b)
    {
        this_thread::sleep_for(chrono::milliseconds(2000));
        cout << "ID potoka = " << this_thread::get_id() << "****t\SUM STARTED\t****" << endl;
        this_thread::sleep_for(chrono::milliseconds(5000));
        cout << "ID potoka = " << this_thread::get_id() << "****t\SUM STOPPED\t****" << endl;
        return a + b;
    }
};

int main()
{
    setlocale(LC_ALL, "ru");
    for (size_t i = 1; i<10; i++)
    {   cout << "ID потока = " << this_thread::get_id() << "\tmain works\t" << i << endl;
        this_thread::sleep_for(chrono::milliseconds(1000));}
    return 0;
}
```

Microsoft Visual Studio Debug Console

```
ID потока = 9536        main works      1
ID потока = 9536        main works      2
ID потока = 9536        main works      3
ID потока = 9536        main works      4
ID потока = 9536        main works      5
ID потока = 9536        main works      6
ID потока = 9536        main works      7
ID потока = 9536        main works      8
ID потока = 9536        main works      9

C:\Users\Admin\source\repos\KN-11 lab1\x64\D
Press any key to close this window . . .
```

Рисунок 1.25 – Додавання трьох методів в клас «MyClass»

Для реалізації методів класу в окремому потоці необхідно створити об’єкт, наприклад «m» класу «MyClass». Для даного об’єкту спочатку використовуємо метод «Sum», в який передамо значення двох чисел, наприклад два та п’ять. Для початку все використовуємо в одному потоці. Також необхідно створити змінну «result», де буде зберігатися результат роботи методу «Sum» (рис. 1.26).

```
int main()
{
    setlocale(LC_ALL, "ru");
    int result;
    MyClass m;
    result = m.Sum(2, 5);
    for (size_t i = 1; i<10; i++)
    {
        cout << "ID потока = " << this_thread::get_id() << "\tmain works\t" << i << endl;
        this_thread::sleep_for(chrono::milliseconds(1000));
    }
    cout << " RESULT \t " << result << endl;
    return 0;
}
```

Рисунок 1.26 – Додавання об’єкту «m» в клас «MyClass»

Наприкінці роботи програми необхідно обов’язково вивести результат на екран (рис. 1.27).

```
Microsoft Visual Studio Debug Console
ID potoka = 14684****tSUM STARTED       ****
ID potoka = 14684****tSUM STOPPED       ****
ID потока = 14684       main works      1
ID потока = 14684       main works      2
ID потока = 14684       main works      3
ID потока = 14684       main works      4
ID потока = 14684       main works      5
ID потока = 14684       main works      6
ID потока = 14684       main works      7
ID потока = 14684       main works      8
ID потока = 14684       main works      9
 RESULT         7

C:\Users\Admin\source\repos\KN-11 lab1\x64\Debug\KN-11 lab1.exe (process 13860) exited with code 0.
Press any key to close this window . . .
```

Рисунок 1.27 – Результат роботи програми

За результатами роботи програми можна зробити висновок, що метод «Sum» потребує виконання в окремому потоці. В потік треба передати анонімну функцію.

У квадратних дужках в анонімній функції треба вказати, з яким змінними с зовнішнього контексту дозволяється працювати лямбда-функції. Передавати можна як за посиланням, так і за значенням, як окремо для кожної змінної, так і для всіх змінних одразу (в прикладі за посиланням зі всіма змінними, тобто вводимо значення «&») (рис. 1.28).

Рисунок 1.28 – Виконання методу «Sum» в окремому потоці

В результаті потік «main» одразу виконується, всередині почала своє виконання функція «Sum» і після невеликої паузи відбувся вивід результату.

Перейдемо до методів «DoWork». Як можна побачити з коду, треба чекати сім секунд до закінчення даного методу, в той час main не виконується (рис. 1.29).

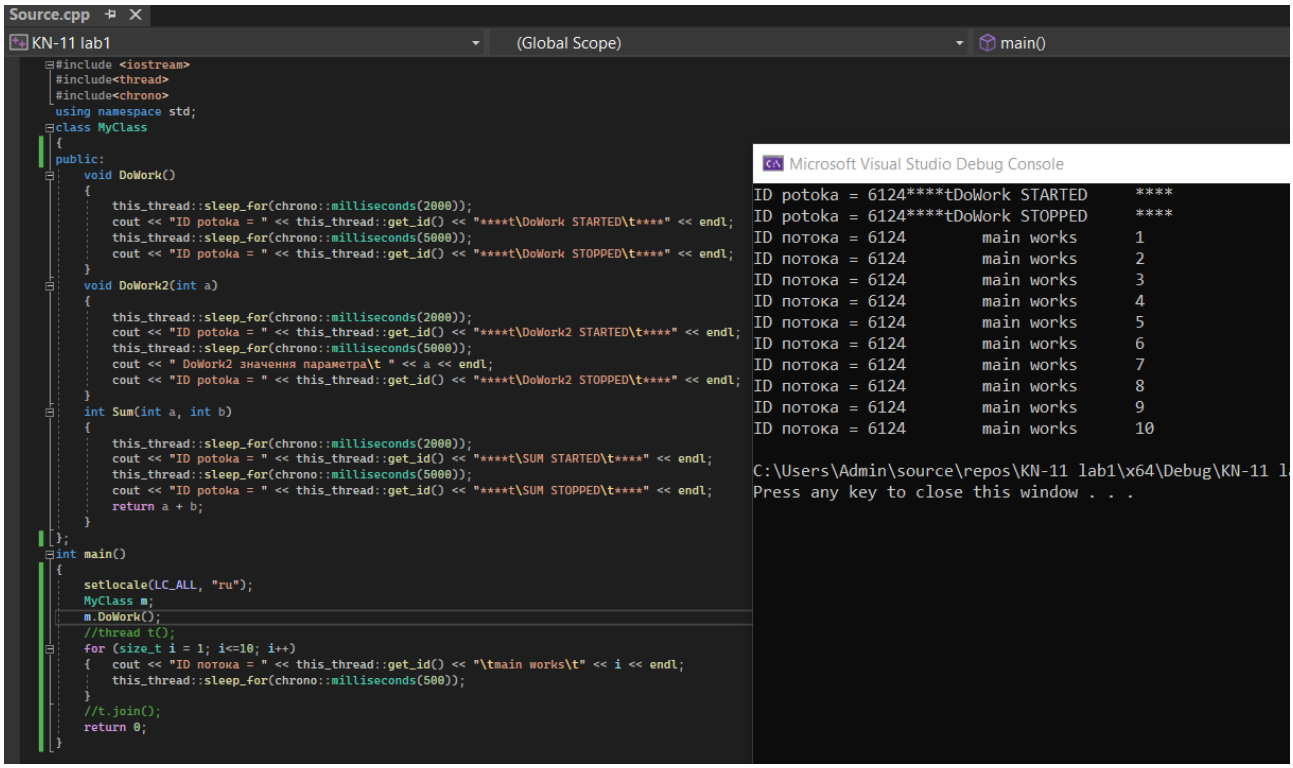

Рисунок 1.29 – Виконання методу «DoWork»

Одним із варіантів роботи потів паралельно є використання лямбда-функцій (рис. 1.30).

Рисунок 1.30 – Виконання методу «DoWork» та main у паралельних потоках

Для методів класу, які не повертають значення, є інший варіант, коли потоку надається посилання на об’єкт класу і через подвійну двокрапку вказати метод, який буде викликатися і у якості другого параметра передати об’єкт класу (рис. 1.31).

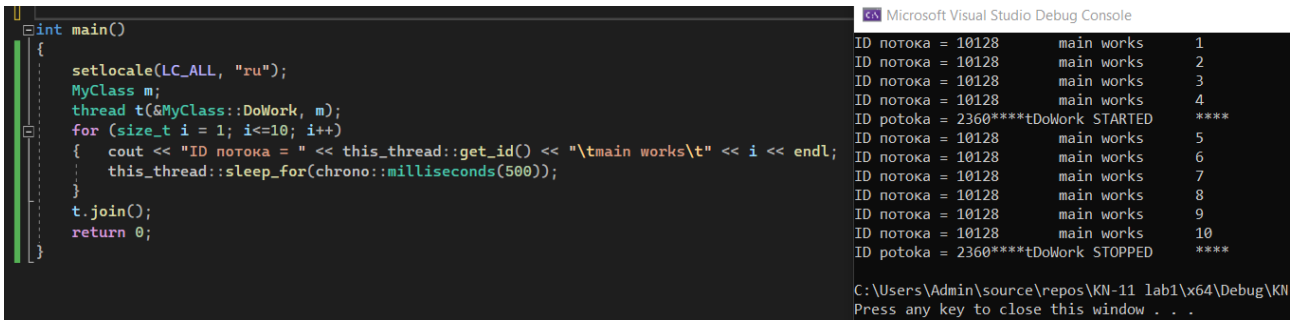

Рисунок 1.31 – Виконання методу «DoWork» та main у паралельних потоках альтернативним варіантом

Розглянемо останній метод «DoWork2», який приймає параметр (на відміну від методу «DoWork»), але не повертає жодних значень (рис. 1.32).

Рисунок 1.32 – Виконання методу «DoWork2» та main у паралельних потоках

Аналогічний результат можна досягти з використанням альтернативної функції.

Запитання для самопідготовки

1. Для чого в C++ створюються класи?
2. Наведіть синтаксис створення класу у C++.
3. Наведіть причину виконання методу «Sum» в окремому потоці.
4. Для якого методи застосовувалась лямбда-функція?

Практичне заняття № 7
ВИКОРИСТАННЯ М'ЮТЕКСІВ

Мета заняття – набути навичок захисту даних, що розділюються та використовуються різними потоками, синхронізації потоків тощо.

Змоделюємо реальну ситуацію. Для прикладу розглянемо гуртожиток університету. В гуртожитку є ресурси, які треба розділяти. Припустимо, що гуртожиток це програма, а здобувачі це потоки в цій програмі. Змоделюємо, що всі здобувачі полюбляють смажену яєчню, але в гуртожитку є лише одна пательня. Таким чином, будь-який студент, у будь-який час може скористатися пательнею. Використання ресурсу, в цьому випадку, нічим не регламентується.

Припустимо, що до пательні підійшло двоє здобувачів та паралельно розбили два яйця на пательню, почали смажитися, пройшов деякий час і підходе третій здобувач і додає, наприклад, ковбасу. Приходить четвертий і починає все солити (перші два також солили). Відбувається повний хаос.

В цьому випадку наш ресурс (пательня) жодним чином не захищений і потоки (здобувачі) не впорядковані. Потоки, при цьому, намагаються змінити дані інших потоків.

Для захисту даних від втручання потоків та синхронізації потоків використовуються м'ютекси. В нашому прикладі в ролі м'ютекса може бути зачинені двері до кухні з пательнею та одним здобувачем.

Двозв’язковий список (ДС) – це структура даних, яка складається з вузлів, які зберігають корисні дані (DATA), покажчики на попередній вузол (pPrev) та наступний вузол (pNext) (рис. 1.33).

Рисунок 1.33 – Схема двозв’язкового списку

Серед контейнерів бібліотеки «STL» є контейнер «list», який використовує модель «двоз’вязкового списку».

Припустимо, що з ДС працює два потоки. Перший потік буде проводити зміни даних в вузлі, а другий потік паралельно зчитувати дані з цього вузла.

Створимо функцію «Print», яка приймає один параметр (змінна типу «char»). За допомогою двох циклів створимо прямокутник (рис. 1.34).

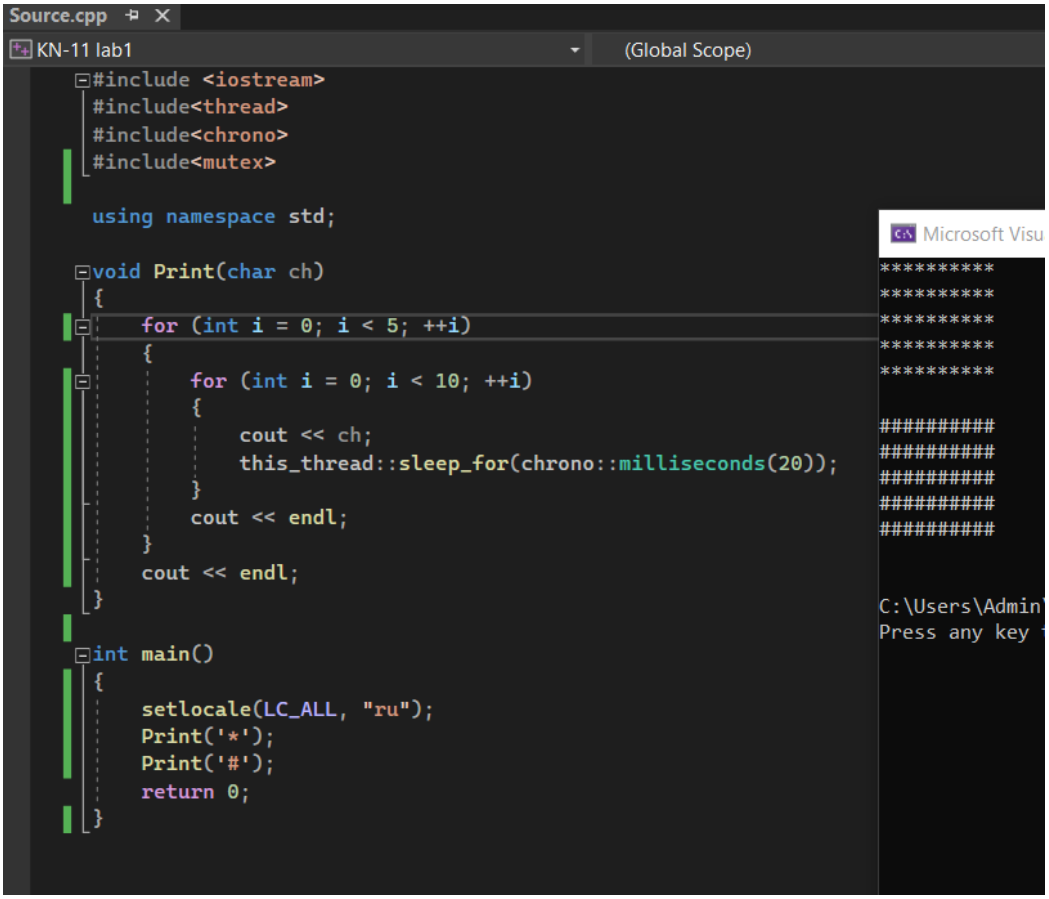

Рисунок 1.34 – Створення двох прямокутників за допомогою циклу

Створимо два потоки (рис. 1.35)

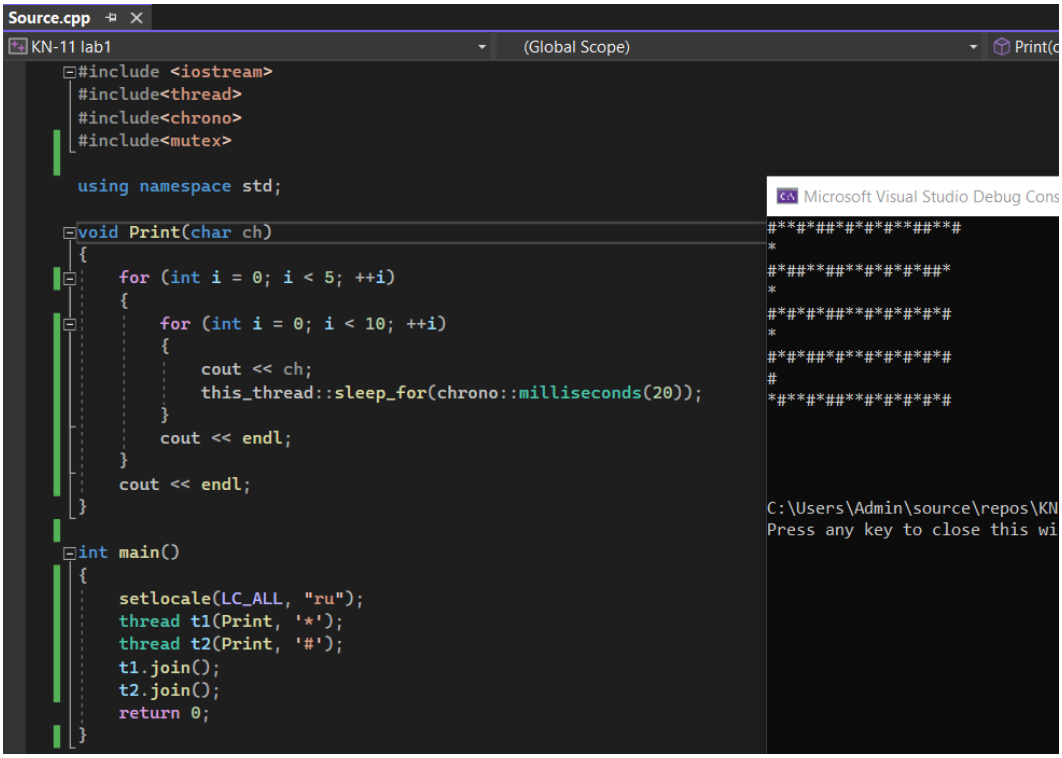

Рисунок 1.35 – Створення двох прямокутників за допомогою двох потоків

Отримуємо небажаний результат. Для синхронізації потоків використаємо м’ютекси. За допомогою м’ютекса необхідно вказати ту секцію в коді, яка повинна працювати лише з одним потоком. У об’єкта «mutex» є два методи «lock» та «unlock» (рис. 1.36).

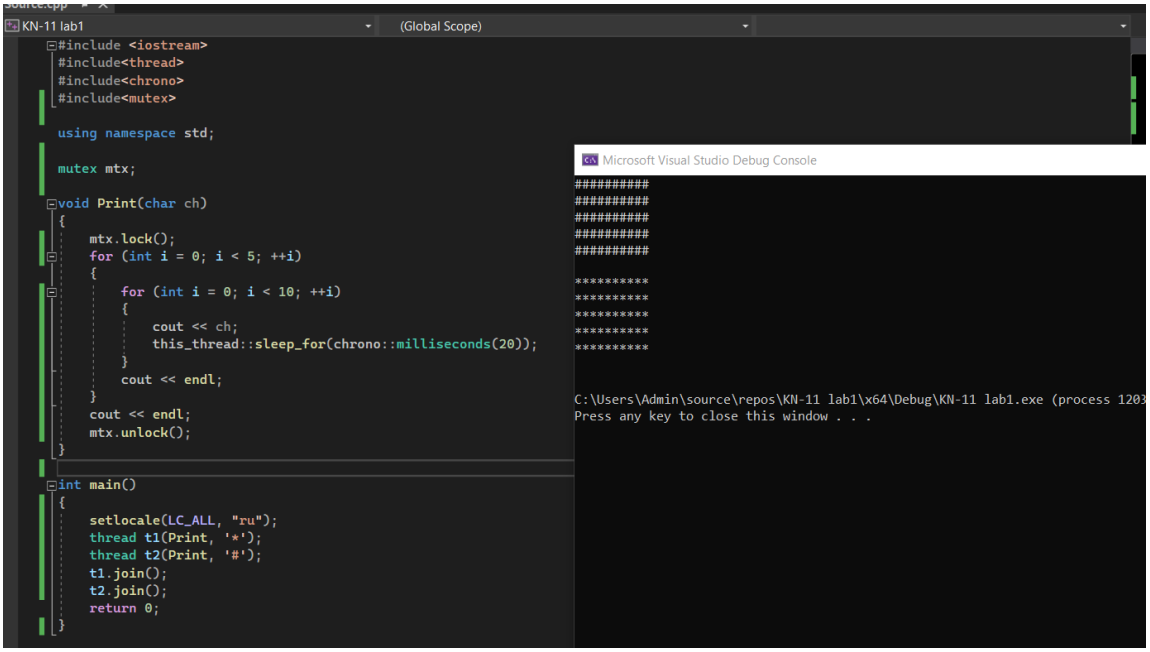

Рисунок 1.36 – Використання м’ютексів

З рисунку 1.36 можна побачити, що відбулась синхронізація двох потоків та був проведений захист консолі від спільного використання, але захист критичних секцій з використанням м’ютексів вимагає більших часових витрат.

Запитання для самопідготовки

1. Яке основне призначення м’ютексів?
2. Яким чином відбувається синхронізація потоків?
3. Чи слід використовувати м’ютекси при створенні транзакцій?
4. Наведіть приклад з реального життя щодо використання м’ютексів.

Практичне заняття № 8
ВИКОРИСТАННЯ КЛАСУ «LOCK_GUARD» ТА ВЗАЄМНЕ БЛОКУВАННЯ

Мета заняття – набути навичок використання класу «lock_guard».

Даний клас використовується для захвату м’ютекса у конструкторі при створенні об’єкту такого класу і звільнити м’ютекс у деструкторі у той момент, коли об’єкт цього класу буде залишати деяку область видимості.

Замість, раніше вивчених методів «lock» та «unlock», можна використовувати lock_guard (рис. 1.37).

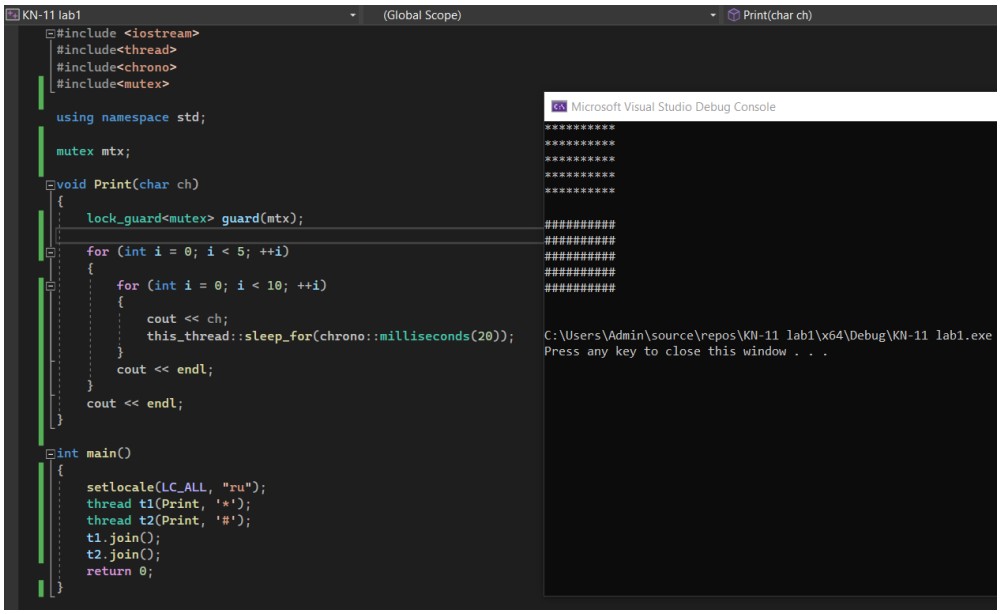

Рисунок 1.37 – Використання класу «lock_guard»

Взаємне блокування реалізуємо в коді з двома м'ютексами і двома функціями, які викликають м'ютекси у різній послідовності (рис. 1.38).

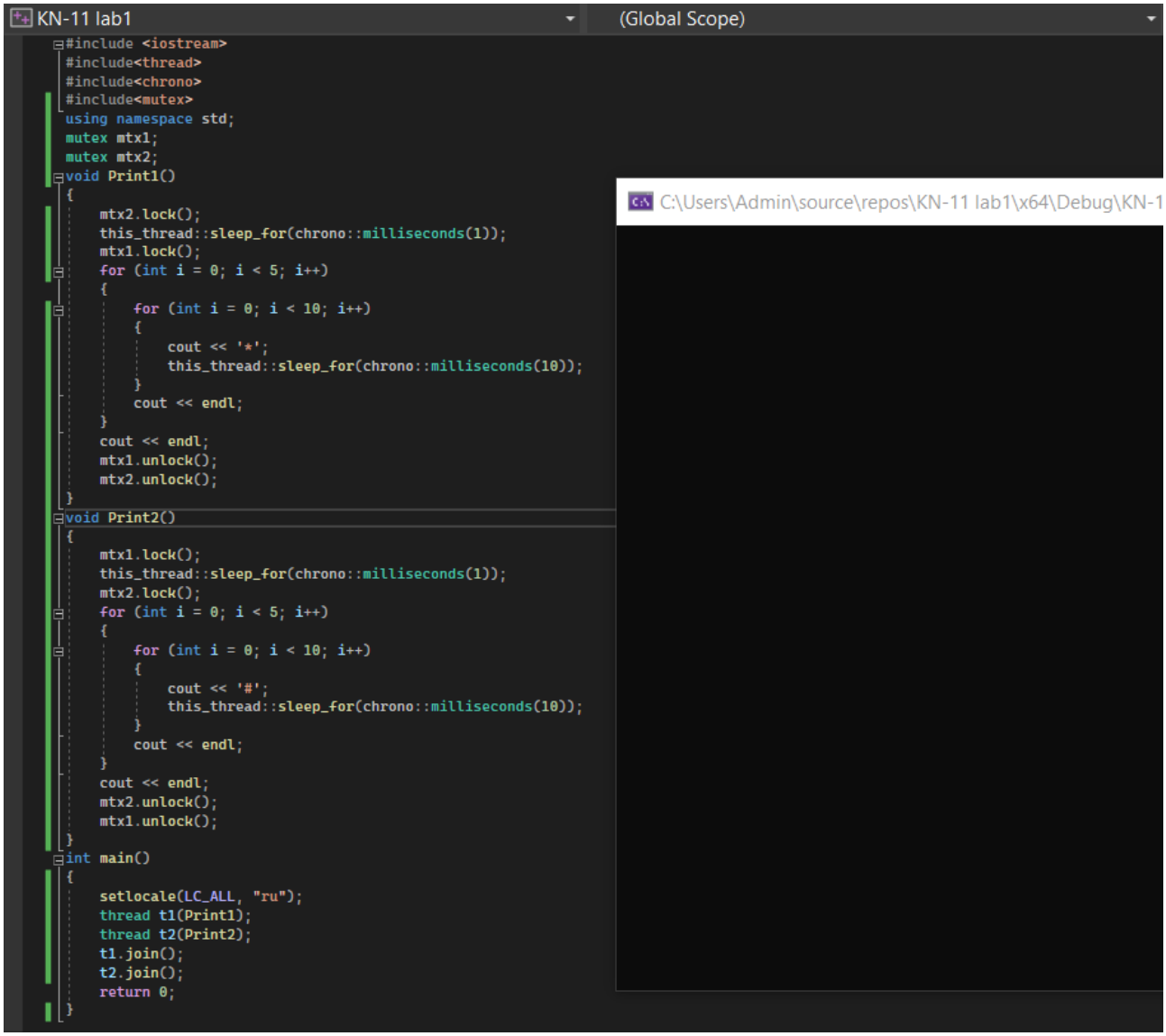

Рисунок 1.38 – Моделювання ситуації взаємного блокування

Запитання для самопідготовки

1. Наведіть синтаксис використання класу «lock_guard».

2. Які методи може замінити клас «lock_guard»?

3. Коли у lock_guard викликається деструктор?

4. У коді, зображеному на рисунку 1.38, виправте помилку взаємного блокування.

2 САМОСТІЙНА РОБОТА

2.1 Загальні рекомендації щодо організації самостійної роботи

Обов'язковим елементом успішного засвоєння навчального матеріалу дисципліни «Технології розподілених систем та паралельних обчислень» є самостійна робота здобувачів вищої освіти з вітчизняною і зарубіжною літературою з питань паралельних та розподілених систем, практики проведення наукових досліджень тощо. Самостійна робота є основним засобом оволодіння навчальним матеріалом у час, вільний від нормованих навчальних занять, тобто лекційних і практичних занять.

Основні види самостійної роботи, на які повинні звертати увагу здобувачі вищої освіти:

– вивчення лекційного матеріалу;

– робота з опрацювання та вивчення рекомендованої літератури;

– підготовка до практичних занять;

– підготовка до обговорень та інших, пропонованих викладачем, завдань;

– робота над рефератами, тезами, доповідями з тематики курсу;

– самоперевірка здобувачем вищої освіти власних знань за запитаннями для самоперевірки;

– підготовка до поточного та підсумкового контролю;

Для цього необхідно:

– розібратися в сутності кожної запропонованої теми;

– підготуватися до дискусії щодо розуміння вивченого матеріалу;

– у разі наявності декількох тлумачень кожного терміну, обґрунтувати, якої саме інтерпретації дотримується здобувач і чому, а також обґрунтувати, з чим здобувач вищої освіти не може погодитись;

– за умови, що значення якогось терміну є незрозумілим, зафіксувати запитання, а під час дискусії за запропонованими темами винести їх на обговорення або проконсультуватися у викладача.

Опрацювання лекційного матеріалу. У системі різних форм навчально-виховної роботи особливе місце належить лекції, де викладач надає здобувачу вищої освіти основну інформацію, навчає розмірковувати, аналізувати, допомагає опанувати ключові знання, а також спрямовує самостійну роботу здобувача.

Зв'язок лекції і самостійної роботи здобувача вищої освіти розглядається в таких напрямах:

– лекція як головна початкова ланка, що визначає зміст і обсяг самостійної роботи здобувача;

– методичні прийоми читання лекцій, що активізують самостійну роботу здобувачів вищої освіти;

– самостійна робота, яка сприяє поглибленому засвоєнню теми на базі прослуханої лекції.

Перший етап самостійної роботи починається з процесу слухання і записування лекції. Правильно складений конспект лекції – найефективніший засіб стимулювання подальшої самостійної роботи здобувачів вищої освіти. Здобувач вищої освіти повинен чітко усвідомити, що конспект лекцій – це короткий тезовий запис головних положень навчального матеріалу.

Конспект допомагає в раціональній підготовці до практичних занять, підсумкового контролю, у визначенні напряму і обсягу подальшої роботи з літературними джерелами. Під час підготовки до лекції здобувач вищої освіти повинен опрацювати матеріал попередньої лекції з використанням підручників та інших джерел літератури.

На лекціях висвітлюють тільки основні теоретичні положення та найбільш актуальні проблеми, тому більшість питань виноситься на самостійне опрацювання.

Підготовка до практичних занять розпочинається з опрацювання лекційного матеріалу. Здобувач вищої освіти повинен самостійно підготувати відповіді на контрольні запитання, які подані в програмі у певній послідовності згідно з логікою засвоєння навчального матеріалу.

Практичні заняття збагачують і закріплюють теоретичні знання здобувачів вищої освіти, розвиваючи їх творчу активність, допомагають у набутті практичних навичок роботи за предметом навчальної дисципліни.

У процесі підготовки до практичних занять самостійна робота здобувачів є обов'язковою частиною навчальної роботи, без якої успішне і якісне засвоєння навчального матеріалу неможливе. Це свідчить про необхідність керування самостійною роботою здобувачів з боку викладача завдяки проведенню цілеспрямованих організаційних і контрольних заходів.

Викладач у вступній лекції рекомендує здобувачам вищої освіти літературу, методичні рекомендації до самостійної роботи та до організації практичних занять з дисципліни. У методичних вказівках з кожної теми наведено перелік питань для теоретичної підготовки до заняття. У разі, коли здобувач не може самостійно розібратися в якомусь питанні, він може отримати консультацію у викладача.

Добре організовані консультації дозволяють спрямувати самостійну роботу в потрібному напрямі, зробити раціональною і підвищити її ефективність.

2.2 Завдання до самостійної роботи

В рамках виконання самостійної роботи необхідно виконати роботу на тему «Розробка паралельних обчислень алгоритму» відповідно до тематики дипломної роботи.

В ході виконання зробити наступні завдання до індивідуальної роботи.

1. Виконати розпаралелювання алгоритму, пояснити обраний спосіб розпаралелювання та обґрунтувати правильність обчислень алгоритму на тестових прикладах.

2. Виконати дослідження часу виконання обчислень для звичайного та алгоритму з розпаралелювання при збільшенні його складності.

3. Побудувати модель паралельних обчислень для оцінки складності алгоритму.

4. Зробити висновки щодо ефективності використання паралельних обчислень та програмного забезпечення, що використовується.

Індивідуальна робота має містити вступ, опис алгоритму, вибір програмного забезпечення для розробки паралельних обчислень та його короткий опис, розробку паралельних обчислень алгоритму з використанням обраної мови програмування (описати стадії проєктування, реалізації та, тестування) та висновки.

СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Коробейнікова Т. І. Комп’ютерні мережі / Т. І. Коробейнікова, С. М. Захарченко. – Львів : Львівська політехніка, 2022. – 228 с.

2. Семеренко В. П. Технології паралельних обчислень : навч. посіб. / В. П. Семеренко. – Вінниця : ВНТУ, 2018. – 104 с.

3. Качко О. Г. Паралельне програмування / О. Г. Качко. – Харків : ХНУРЕ, 2016. – 403 с.

4. Малашонок Г. І. Паралельні обчислення на розподіленій пам’яті : OpenMPI, Java, Math Partner / Г. І. Малашонок, А. А. Сідько. – Київ : НаУКМА, 2020. – 266 с.

5. Луцків А. М. Паралельні та розподілені обчислення / А. М. Луцків, С. А. Лупенко, В. В. Пасічник. – Львів : Магнолія, 2021. – 168 с.

Електронне навчальне видання

Методичні рекомендації
до проведення практичних занять та організації самостійної роботи
з навчальної дисципліни

«КОМП'ЮТЕРНІ МЕРЕЖІ»

(для здобувачів першого (бакалаврського) рівня вищої освіти
денної та заочної форм навчання
зі спеціальності 122 – Комп'ютерні науки,
освітньо-професійна програма «Комп'ютерні науки»)

Укладач **ПЛАХОТНІКОВ** Кирило Валерійович

Відповідальний за випуск *М. В. Новожилова*
За авторською редакцією
Комп'ютерне верстання *К. В. Плахотніков*

План 2023, поз. 232М

Підп. до друку 31.03.2023. Формат 60 × 84/16.
Ум. друк. арк. 2,5.

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Маршала Бажанова 17, Харків, 61002.
Електронна адреса: office@kname.edu.ua
Свідоцтво суб'єкта видавничої справи:
№ ДК 5328 від 11.04.2017.