Міністерство освіти і науки України
Національний технічний університет
України
«Київський політехнічний інститут»

**Лабораторна робота №4**

# ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ ЗА ДОПОМОГОЮ STL

Виконав студент групи: КВ-03

ПІБ: Жовтанюк М.В

**Київ 2022**

**Паралельне програмування за допомогою STL**

Завдання:

**Частина 1**

Реалізувати паралельну обробку елементів масиву відповідно до варіанту, використовуючи механізм асинхронних викликів (std::async). Для цього розбити масив на M однакових частин та виконувати обробку кожної з них
окремо. Масив початково заповнюється випадковими значеннями.
Виміряти час обробки масиву для декількох різних значень M (наприклад,
1, 2, 3, 4, 8, 16), а також порівняти його для випадків послідовної та паралельної обробки (використовувати політики запуску std::launch::async та
std::launch::deferred). Результати дослідження представити у вигляді таблиці у
звіті. Бажано також у звіті представити короткі висновки з аналізу отриманих
результатів.
Розмір масиву обрати на власний розсуд в залежності від конфігурації апаратного забезпечення системи (орієнтуватися на час роботи не менший
ніж 10 мс для забезпечення точності вимірів). Тип елементів масиву обрати
також самостійно.
В якості додаткового завдання – реалізувати можливість задання розміру масиву, кількості частин (для розбиття) та політики запуску користувачем.
Номер варіанту визначається відповідно до номеру за списком за наступною формулою:

**Варіанти завдань:**

2. Розрахунок суми значень елементів масиву
константне значення (з перезаписом значень масиву)

**Частина 2**

Реалізувати систему, що зображена на рисунку.

Система містить наступні сутності: N «генераторів» та один «процесор». Генератор генерує випадкове значення та передає його процесору, після чого «засинає» на випадковий проміжок часу (діапазон значень обрати самостійно). Процесор чекає значення від генераторів та виводить їх на екран (виводячи також ідентифікатор (наприклад, порядковий номер) генератора). Кожна з сутностей (як генератори, так і процесор) має виконуватися в окремому потоці.

Реалізувати механізм завершення роботи програми (наприклад, шляхом натиснення деякої клавіші).

В якості додаткового завдання може бути реалізовано, наприклад, процедури додавання/видалення генераторів, а також конфігурування кожного з них (діапазон часу затримки, діапазон генерованих значень).

Код програми:
main.cpp

```
#include <iostream>
#include "async.h"

void part_1();
void part_2();

std::vector<long> async_vector;
std::vector<std::future<long>> res_vector;
int VECTOR_SIZE = 100000000;
int parts_amount = 0;
int l_policy_option = 0;

using namespace std::chrono;

int main() {
    int option = 2;
    std::cout << "1. First part, time measurement" << std::endl;
    std::cout << "2. Second part, processor-generator" << std::endl;
    std::cout << '\n';

    switch (option) {
        case 1:
            part_1();
            break;
        case 2:
            part_2();
            break;
        default:
            break;
    }
    return 0;
}

void part_1() {

    std::cout << "Enter array size: ";
    std::cin >> VECTOR_SIZE;
    std::cout << '\n';

    std::launch l_policy;

    std::cout << "Please enter amount of parts: ";
    std::cin >> parts_amount;

    std::cout << "Press 1 to launch in deferred policy or 2 for async: ";
    std::cin >> l_policy_option;

    if (l_policy_option == 1) l_policy = std::launch::deferred;
    else if (l_policy_option == 2) l_policy = std::launch::async;
    else {
        std::cout << "Wrong option, terminating..." << std::endl;
        exit(1);
    }

    async_vector.resize(VECTOR_SIZE);
    for (auto &i: async_vector) {
        i = random() % VECTOR_SIZE;
    }

    std::vector<std::vector<long>> parts;
    parts.resize(parts_amount);
```

```
    int one_part_amount = VECTOR_SIZE / parts_amount;
    for (int i = 0, j = 0; i < async_vector.size(); i++) {
        parts[j].push_back(async_vector[i]);
        if (i == one_part_amount) j++;
    }

    // async
    for (int i = 0; i < parts.size(); i++) {
        res_vector.push_back(std::async(l_policy, sum_calc, parts[i], i));
    }

    long res_sum = 0;

    // start
    auto start = steady_clock::now();

    for (auto &i: res_vector) {
        res_sum += i.get();
    }

    auto stop = steady_clock::now();
    // stop


    auto time_res = duration_cast<std::chrono::milliseconds>(stop - start);
    auto ms = time_res.count();

    std::cout << "Time: " << ms << std::endl;
    std::cout << "Sum result: " << res_sum << std::endl;
}

void part_2() {
    std::thread th_proc(processor);
    th_proc.join();
}
```

## async.h

```
//
// Created by Maksym Zhovtaniuk on 19.06.2022.
//
#ifndef OOP_LAB4_ASYNC_H
#define OOP_LAB4_ASYNC_H

#include <future>
#include <vector>
#include <cstdlib>
#include <string>
#include <chrono>
#include <sstream>

std::mutex m;
static bool exit_status;

long sum_calc(const std::vector<long> &v, const int tag) {
    long res = 0;
    for (auto &i: v) {
        res += i;
    }
    return res;
}

void generator(std::vector<std::vector<long>> &v, const int tag) {
    while (!exit_status) {
        m.lock(); // lock mutex

        long random_num = random();
        long ms = 1000 + random() % (5000 - 1000 + 1);

        std::ostringstream oss;
        oss << "Thread" << tag << " returns random number = " << random_num << "
and sleeps for " << ms
            << " milliseconds";
        std::string str = oss.str();

        std::cout << str << std::endl;

        std::vector<long> res = {tag, random_num, ms};
        v.push_back(res);

        m.unlock(); // unlock mutex
        std::this_thread::sleep_for(std::chrono::milliseconds(ms));
    }
}

void processor() {
    exit_status = false;

    int generators_num = 3;

    std::vector<std::thread> threads_generators;
    threads_generators.resize(generators_num);
    std::vector<std::vector<long>> random_res;

    for (int i = 0; i < generators_num; i++) {
        threads_generators[i] = std::thread(&generator, std::ref(random_res), i);
    }

    std::cin.get();
    exit_status = true;

    for (auto &i: threads_generators) {
        i.join();
```

```
    }

    std::cout << "All threads has been finished" << std::endl;
    std::cin.get();
}

#endif //OOP_LAB4_ASYNC_H
```

- Посилання на github-репозиторій
- Також у проекті використано CMake

Принцип роботи:

1. Користувач задає кількість частин та розмір масиву, далі цей вектор заповнюється випадковими значеннями у діапазоні [0, розмір_масиву], створюється вектор векторів з довжиною N – кількість частин (по суті матриця), кожен вкладений вектор буде передаватися як параметр у потік, елементи цього масиву заповнюється відповідними значеннями із початкового. Створюється вектор ф’ючерсів, де параметрами у std::async передаються: політика виклику (послідовна чи асинхронна), вказівник на функцію, частина, яку буде обробляти потік та порядковий номер потоку. Далі проводяться заміри часу та виконуються асинхронні виклики.

2. Створюється потік-процесор, в якому користувач спочатку вказує кількість генераторів, потім створюється вектор потоків та вектор результатів, який містить: номер потоку, випадкове число, що потік створить та випадковий час затримки після кожної ітерації, далі створюються потоки, у які передаються: функція, що виконуватиметься, вказівник на вектор, у який будуть заноситись результати кожної ітерації та номер потоку. Процесор чекає поки користувач натисне ENTER, а потім закінчує потоки. Генератори-потоки працюють поки користувач не вийде (використовується булева змінна, яка міняє стан по натисненні на ENTER), спочатку блокується раніше ініціалізований м’ютекс, щоб ніхто не «вліз» у роботу потока, генерується випадкове число та випадкове значення затримки в діапазоні [1000, 5000] (від 1 до 5 секунд), потік виводить результати та заносить у вектор результатів значення кожної ітерації, далі м’ютекс розблоковується і потік «засинає» на визначену раніше випадкову величину.

**Тести:**

**Частина 1:**

```
Enter array size: 100000000
Please enter amount of parts: 1
Press 1 to launch in deferred policy or 2 for async: 2
Time: 1007
Sum result: 4942469835882961
Enter array size: 100000000
Please enter amount of parts: 10
Press 1 to launch in deferred policy or 2 for async: 1
Time: 993
Sum result: 4942469835882961
Enter array size: 100000000
Please enter amount of parts: 10
Press 1 to launch in deferred policy or 2 for async: 2
Time: 932
Sum result: 4942469835882961
Enter array size: 10000000
Please enter amount of parts: 10
Press 1 to launch in deferred policy or 2 for async: 1
Time: 104
Sum result: 49952491479754
Enter array size: 10000000
Please enter amount of parts: 10
Press 1 to launch in deferred policy or 2 for async: 2
Time: 93
Sum result: 49952491479754
Enter array size: 10000000
Please enter amount of parts: 5
Press 1 to launch in deferred policy or 2 for async: 1
Time: 96
Sum result: 49952491479754
Enter array size: 10000000
Please enter amount of parts: 5
Press 1 to launch in deferred policy or 2 for async: 2
Time: 77
Sum result: 49952491479754
```

```
Enter array size: 10000000
Please enter amount of parts: 1
Press 1 to launch in deferred policy or 2 for async: 1
Time: 101
Sum result: 49952491479754

Enter array size: 10000000
Please enter amount of parts: 1
Press 1 to launch in deferred policy or 2 for async: 2
Time: 102
Sum result: 49952491479754
```

*Усі тести нижче розміру масиву 1000000 дають нуль

Кількість частин M = 10

| **Розмір масиву** | **Deferred (послідовний)** | **Async (асинхронний)** |
|---|---|---|
| 1000000 | 10 | 9 |
| 10000000 | 96 | 92 |
| 100000000 | 1080 | 911 |

Кількість частин M = 5

| **Розмір масиву** | **Deferred (послідовний)** | **Async (асинхронний)** |
|---|---|---|
| 1000000 | 10 | 8 |
| 10000000 | 95 | 77 |
| 100000000 | 999 | 819 |

Кількість частин M = 1

| **Розмір масиву** | **Deferred (послідовний)** | **Async (асинхронний)** |
|---|---|---|
| 1000000 | 10 | 10 |
| 10000000 | 96 | 101 |
| 100000000 | 1012 | 1007 |

**Висновок:**

У загальному випадку видно, що зі збільшенням розміру масиву, швидкість роботи асинхронного режиму зменшується час виконання задачі обчислення суми, зменшення ж однакових частин масиву, поводить себе непередбачувано, то збільшуючи, то зменшуючи час роботи.

**Частина 2:**

```
Thread0 returns random number = 1804289383 and sleeps for 4207 milliseconds
Thread1 returns random number = 1681692777 and sleeps for 1363 milliseconds
Thread2 returns random number = 1957747793 and sleeps for 1302 milliseconds
Thread3 returns random number = 719885386 and sleeps for 1155 milliseconds
Thread4 returns random number = 596516649 and sleeps for 1085 milliseconds
Thread4 returns random number = 1025202362 and sleeps for 1489 milliseconds
Thread3 returns random number = 783368690 and sleeps for 1498 milliseconds
Thread2 returns random number = 2044897763 and sleeps for 3171 milliseconds
Thread1 returns random number = 1365180540 and sleeps for 3427 milliseconds
Thread4 returns random number = 304089172 and sleeps for 2954 milliseconds
Thread3 returns random number = 35005211 and sleeps for 2002 milliseconds
Thread0 returns random number = 294702567 and sleeps for 1798 milliseconds
Thread2 returns random number = 336465782 and sleeps for 3329 milliseconds
Thread3 returns random number = 278722862 and sleeps for 3722 milliseconds
Thread1 returns random number = 2145174067 and sleeps for 2989 milliseconds
Thread4 returns random number = 1101513929 and sleeps for 2420 milliseconds
Thread0 returns random number = 1315634022 and sleeps for 1167 milliseconds
Thread0 returns random number = 1369133069 and sleeps for 1763 milliseconds
Thread1 returns random number = 1059961393 and sleeps for 1332 milliseconds
Thread2 returns random number = 628175011 and sleeps for 1026 milliseconds
Thread4 returns random number = 1131176229 and sleeps for 1132 milliseconds
Thread3 returns random number = 859484421 and sleeps for 3403 milliseconds
Thread2 returns random number = 608413784 and sleeps for 2360 milliseconds
Thread0 returns random number = 1734575198 and sleeps for 2049 milliseconds
Thread4 returns random number = 149798315 and sleeps for 3832 milliseconds
Thread1 returns random number = 1129566413 and sleeps for 2337 milliseconds


All threads has been finished
```

```
Thread0 returns random number = 1804289383 and sleeps for 4207 milliseconds
Thread1 returns random number = 1681692777 and sleeps for 1363 milliseconds
Thread3 returns random number = 1957747793 and sleeps for 1302 milliseconds
Thread2 returns random number = 719885386 and sleeps for 1155 milliseconds
Thread4 returns random number = 596516649 and sleeps for 1085 milliseconds
Thread4 returns random number = 1025202362 and sleeps for 1489 milliseconds
Thread2 returns random number = 783368690 and sleeps for 1498 milliseconds
Thread3 returns random number = 2044897763 and sleeps for 3171 milliseconds
Thread1 returns random number = 1365180540 and sleeps for 3427 milliseconds
Thread4 returns random number = 304089172 and sleeps for 2954 milliseconds
Thread2 returns random number = 35005211 and sleeps for 2002 milliseconds
Thread0 returns random number = 294702567 and sleeps for 1798 milliseconds
Thread3 returns random number = 336465782 and sleeps for 3329 milliseconds
Thread2 returns random number = 278722862 and sleeps for 3722 milliseconds
Thread1 returns random number = 2145174067 and sleeps for 2989 milliseconds
Thread4 returns random number = 1101513929 and sleeps for 2420 milliseconds
Thread0 returns random number = 1315634022 and sleeps for 1167 milliseconds
Thread0 returns random number = 1369133069 and sleeps for 1763 milliseconds
Thread1 returns random number = 1059961393 and sleeps for 1332 milliseconds
Thread3 returns random number = 628175011 and sleeps for 1026 milliseconds
Thread4 returns random number = 1131176229 and sleeps for 1132 milliseconds
Thread2 returns random number = 859484421 and sleeps for 3403 milliseconds
Thread3 returns random number = 608413784 and sleeps for 2360 milliseconds
Thread0 returns random number = 1734575198 and sleeps for 2049 milliseconds
Thread4 returns random number = 149798315 and sleeps for 3832 milliseconds
Thread1 returns random number = 1129566413 and sleeps for 2337 milliseconds

All threads has been finished
Thread0 returns random number = 1804289383 and sleeps for 4207 milliseconds
Thread2 returns random number = 1681692777 and sleeps for 1363 milliseconds
Thread1 returns random number = 1957747793 and sleeps for 1302 milliseconds
Thread4 returns random number = 719885386 and sleeps for 1155 milliseconds
Thread3 returns random number = 596516649 and sleeps for 1085 milliseconds
Thread3 returns random number = 1025202362 and sleeps for 1489 milliseconds
Thread4 returns random number = 783368690 and sleeps for 1498 milliseconds
Thread1 returns random number = 2044897763 and sleeps for 3171 milliseconds
Thread2 returns random number = 1365180540 and sleeps for 3427 milliseconds
Thread3 returns random number = 304089172 and sleeps for 2954 milliseconds
Thread4 returns random number = 35005211 and sleeps for 2002 milliseconds
Thread0 returns random number = 294702567 and sleeps for 1798 milliseconds

All threads has been finished
```