ФГАОУ ВО Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа киберфизических систем и управления

**ОТЧЕТ**

**по лабораторной работе №3**

**по теме «Многопоточность»**

по дисциплине «Практикум по программированию»

Выполнил

студент гр.23533/2 А.Д. Шурак

Проверил

ассистент В.Э. Ковалевский

Санкт-Петербург

2019

Задание

На основе полученных знаний продемонстрировать реализацию такого свойства, как многопоточность.

Разработка программы

Реализуем программу, которая будет заполнять массивы рандомными числами и параллельно выполнять сортировку одинаковых массивов.

1. Подключаем необходимые библиотеки.

#include <thread>//библиотека для мультипоточности

#include <chrono>//библиотека для таймера

#include <mutex>//библиотека, чтобы блокировать

1. Реализуем сортировки массивов, используя быструю сортировку и сортировку пузырьком.

void quicksort(int \*array, int left, int right) – сортировка «быстрая»

{

int i = left;

int j = right;

int pivot = array[(i + j) / 2];

int temp;

while (i <= j)

{

while (array[i] < pivot)

i++;

while (array[j] > pivot)

j--;

if (i <= j)

{

temp = array[i];

array[i] = array[j];

array[j] = temp;

i++;

j--;

}

}

if (j > left)

quicksort(array, left, j);

if (i < right)

quicksort(array, i, right);

}

void bubblesort(int \*array, int size) – сортировка «пузырьком»

{

int tmp;

for (int i = 0; i < size - 1; ++i)

{

for (int j = 0; j < size - 1; ++j)

{

if (array[j + 1] < array[j])

{

tmp = array[j + 1];

array[j + 1] = array[j];

array[j] = tmp;

}

}

}

}

1. Пропишем главную функцию, в которой будут созданы потоки и заполнение массива рандомными числами.

Создаём рандомный генератор для заполнения массива:

default\_random\_engine generator;

uniform\_int\_distribution<int> distribution(1, 1000);

int size;

cout << "Enter the size of the arrays:\n";

cin >> size;

Засекаем время для выполнения программы после ввода размера массива:

auto start = chrono::high\_resolution\_clock::now();

int \*array1 = new int[size];

int \*array2 = new int[size];

Засекаем время для рандомайзера:

auto start\_rnd = chrono::high\_resolution\_clock::now();

for (int i = 0; i < size; i++)

{

array1[i] = distribution(generator);

array2[i] = array1[i];

}

auto end\_rnd = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration\_rnd = end\_rnd - start\_rnd;

cout << "Duration of the Randomizer: " << duration\_rnd.count() << endl;

Создаем потоки для сортировок:

thread quick\_thread(quicksort\_time, array1, 0, (size - 1));

thread bubble\_thread(bubblesort\_time, array2, size);

Присоединяем потоки, чтобы они не ушли, когда заканчивается основная функция:

if (quick\_thread.joinable())

quick\_thread.join();

if (bubble\_thread.joinable())

bubble\_thread.join();

delete[] array1;

delete[] array2;

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration = end - start;

cout << "Duration of the Main Function: " << duration.count() << endl;

system("pause");

return 0;

И заканчиваем функцию.

1. В самом начале инициализируем объект, который будет блокировать доступ к чему-то для всех, кроме одного потока сортировки.

mutex g\_lock;

1. Напишем таймеры сортировок.

void quicksort\_time(int \*array, int left, int right)

{

auto start = chrono::high\_resolution\_clock::now();

quicksort(array, left, right);

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration = end - start;

showtime(duration, "Quick Sort");

}

void bubblesort\_time(int \*array, int size)

{

auto start = chrono::high\_resolution\_clock::now();

bubblesort(array, size);

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration = end - start;

showtime(duration, "Bubble Sort");

}

1. Реализуем вывода таймера на консоль. Он общий для для того, чтобы два потока не начали записывать в консоль одновременно. Поэтому блокируется через созданный в начале объект.

void showtime(chrono::duration<float> time, const char\* namesort)

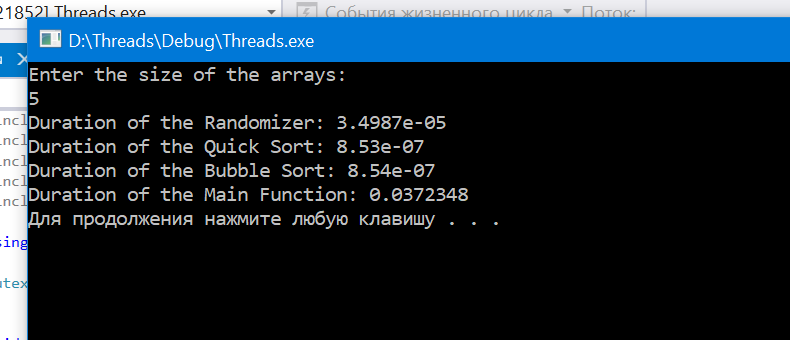
{

g\_lock.lock();

cout << "Duration of the " << namesort << ": " << time.count() << endl;

g\_lock.unlock();

}

**Демонстрация работы программ****ы**

**Заключение**

В ходе выполнения лабораторной работы был продемонстрирован такой процесс, как многопоточность.

**Приложение**

#include <iostream>

#include <random>

#include <thread>//библиотека для мультипоточности

#include <chrono>//библиотека для таймера

#include <mutex>//библиотека, чтобы блокировать

using namespace std;

mutex g\_lock;//объект, чтобы заблокировать доступ к чему-то для всех, кроме одного потока

//сортировки

void quicksort(int \*array, int left, int right)

{

int i = left;

int j = right;

int pivot = array[(i + j) / 2];

int temp;

while (i <= j)

{

while (array[i] < pivot)

i++;

while (array[j] > pivot)

j--;

if (i <= j)

{

temp = array[i];

array[i] = array[j];

array[j] = temp;

i++;

j--;

}

}

if (j > left)

quicksort(array, left, j);

if (i < right)

quicksort(array, i, right);

}

void bubblesort(int \*array, int size)

{

int tmp;

for (int i = 0; i < size - 1; ++i)

{

for (int j = 0; j < size - 1; ++j)

{

if (array[j + 1] < array[j])

{

tmp = array[j + 1];

array[j + 1] = array[j];

array[j] = tmp;

}

}

}

}

/\*вывод таймера в консоль. Общий для того, чтобы два потока не полезли записывать в консоль одновременно

Поэтому блокируется через созданный в начале объект\*/

void showtime(chrono::duration<float> time, const char\* namesort)

{

g\_lock.lock();

cout << "Duration of the " << namesort << ": " << time.count() << endl;

g\_lock.unlock();

}

//таймеры сортировок

void quicksort\_time(int \*array, int left, int right)

{

auto start = chrono::high\_resolution\_clock::now();

quicksort(array, left, right);

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration = end - start;

showtime(duration, "Quick Sort");

}

void bubblesort\_time(int \*array, int size)

{

auto start = chrono::high\_resolution\_clock::now();

bubblesort(array, size);

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration = end - start;

showtime(duration, "Bubble Sort");

}

int main()

{

//тут создается рандомный генератор для заполнения массивов

default\_random\_engine generator;

uniform\_int\_distribution<int> distribution(1, 1000);

int size;

cout << "Enter the size of the arrays:\n";

cin >> size;

//засекается время для выполнения программы после ввода размера массивов

auto start = chrono::high\_resolution\_clock::now();

int \*array1 = new int[size];

int \*array2 = new int[size];

//засекается время для рандомайзера, можно убрать

auto start\_rnd = chrono::high\_resolution\_clock::now();

for (int i = 0; i < size; i++)

{

array1[i] = distribution(generator);

array2[i] = array1[i];

}

auto end\_rnd = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration\_rnd = end\_rnd - start\_rnd;

cout << "Duration of the Randomizer: " << duration\_rnd.count() << endl;

//создаются потоки для сортировок

thread quick\_thread(quicksort\_time, array1, 0, (size - 1));

thread bubble\_thread(bubblesort\_time, array2, size);

//присоединяем потоки, чтобы они не уплыли, когда основная функция закончится

if (quick\_thread.joinable())

quick\_thread.join();

if (bubble\_thread.joinable())

bubble\_thread.join();

delete[] array1;

delete[] array2;

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration = end - start;

cout << "Duration of the Main Function: " << duration.count() << endl;

system("pause");

return 0;

}