МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчет по лабораторной №3

по дисциплине «Системное программирование»

**Многопоточность в языке C++**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил: | студент гр. ФИБ-3301-51-01 |  | / В. Р. Кочкин / |
| Проверил: | к.ф.-м.н. доцент каф. ПМиИ |  | / В. А. Бызов / |

Киров 2022

Цель работы

Получить навыки работы с потоками в языке C++.

Задания

**Задание 0**

Запустить на выполнение пример 1. Заменить join на detach.

Прокомментировать результат.

**Полученные результаты**

При отсоединении дочернего потока он не успеваем выводить свою информацию, так как главный поток завершается быстрее.

Листинг программы приведен в [приложении А задание 0](#_Задание_0.).

**Задание 1**

Разработать программу, выводящую числа от 1 до 100 в двух потоках, в первом – чётные, во втором – нечётные. Запустить несколько раз на выполнение.

Вопрос. В каком порядке будут выводиться числа в консоли? Почему?

**Полученные результаты**

Изображение выглядит как стол

Автоматически созданное описание

Потоки выводят свою информацию беспорядочно, потому что они не синхронизированы.

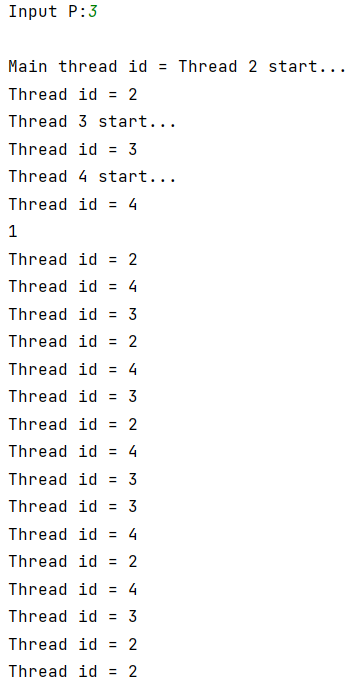
Листинг программы приведен в [приложении А задание 1](#_Задание_1.).

**Задание 2**

Разработать программу, генерирующую P потоков (число P <= 10 задаётся при запуске), каждый из которых сначала выводит сообщение о старте, а потом выводит свой номер 100 раз. Количество потоков P задает пользователь с клавиатуры. Не забудьте имитировать сложные вычисления в потоке.

Длительность сложных вычислений – случайное число миллисекунд из диапазона [1000, 2000].

**Полученные результаты**



Листинг программы приведен в [приложении А задание 2](#_Задание_2.).

**Задание 3**

Создать P потоков (число P <= 10 задаётся при запуске), конкурирующих за общий ресурс – целочисленную переменную. Каждый поток увеличивает значение переменной на свой индекс. Потоки должны завершиться после того, как значение общей переменной превысило 100 (начальное значение – 0).

Организовать корректное взаимодействие потоков с использованием объекта std::mutex.

**Полученные результаты**

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Листинг программы приведен в [приложении А задание 3](#_Задание_3.).

**Задание 4**

Написать программу, которая каждый элемент массива размера N заменяет на его наибольший простой делитель. Число N задается пользователем. Элементы массива – случайные натуральные числа из диапазона [105, 106]. Распараллелить с использованием std::thread. Замерить время работы программы для N = 2·107, 5·107 и 108 на 1, 2, 4 и 8 потоках. Для проверки результатов реализовать данную задачу последовательно, сравнить результаты.

На каждом примере запустить не менее 3 раз. В таблицу занести среднее время выполнения на одном примере в секундах.

Замечание. Если N не делится на число потоков, то нужно очень аккуратно раздавать нагрузку! Проверяйте, что у Вас два потока одновременно не делают одно и то же, также необходимо проверить, что нет «отдыхающих» потоков.

Вычислить ускорения для каждого значения N (заполнить таблицу 2) и построить диаграмму зависимости ускорения от числа потоков (три графика на одной диаграмме).

Замечание. Ускорение вычисляется как отношение времени работы последовательной программы к времени работы параллельной программы.

**Полученные результаты**

Изображение выглядит как текст

Автоматически созданное описание

*Таблица 1 – Время обработки массивов, с*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размерность | Число потоков | | | |
| 1 | 2 | 4 | 8 |
| 107 | 1,65938 | 0,929181 | 0,516937 | 0,325295 |
| 108 | 16,673 | 8,91569 | 5,09134 | 3,18233 |
| 109 | 167,229 | 91,4021 | 49,1689 | 30,8078 |

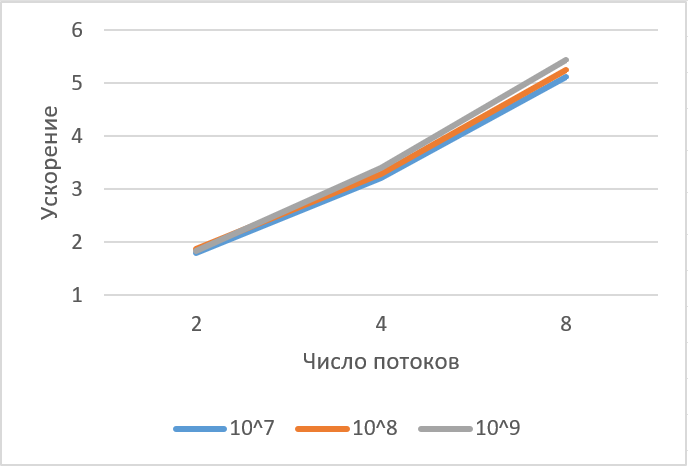
Из таблицы видно, что программа быстрее всего исполнялась на 8 потоках.

*Таблица 2 – Ускорение параллельного суммирования массивов, с*

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность | Число потоков | | |
| 2 | 4 | 8 |
| 107 | 1,79 | 3,210024 | 5,10 |
| 108 | 1,87 | 3,274776 | 5,24 |
| 109 | 1,83 | 3,401113 | 5,43 |

Из таблицы видно, что максимальное ускорение при 8 потоках, однако ускорение не достигает 8.

*График 1 – Ускорение параллельного суммирования массивов, с*



Листинг программы приведен в [приложении А задание 4](#_Задание_4.).

**Задание 5**

Разработать программу, включающую в себя последовательный и параллельный алгоритм вычисления произведения квадратной матрицы на вектор. Каждый алгоритм реализуется в отдельном методе. Матрица и вектор генерируются некоторым способом по заданной размерности N. Параллельный алгоритм должен учитывать доступное ему количество процессоров. Программа запрашивает у пользователя размерность N, после чего выводит время вычислений для последовательного и параллельного алгоритмов. Сами матрицы выводить не нужно.

Провести тестирование программ на матрицах размерности N = 5000, 10000 и 20000. На каждом примере запустить не менее 3 раз. В таблицы занести среднее время выполнения на одном примере в секундах и ускорение. Построить диаграмму зависимости ускорения от числа потоков (три графика на одной диаграмме).

**Полученные результаты**

*Проверка работы алгоритма.*

Изображение выглядит как текст

Автоматически созданное описание

*Таблица 3 – Время выполнения алгоритма умножения*

*матрицы на вектор, с*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размерность | Алгоритм | | | |
| Последовательный | Параллельный | | |
| 2 | 4 | 8 |
| 5000 | 0,128439 | 0,0655188 | 0,0356947 | 0,0245907 |
| 10000 | 0,606812 | 0,309391 | 0,173914 | 0,104315 |
| 20000 | 3,30509 | 1,6949 | 0,930918 | 0,524831 |

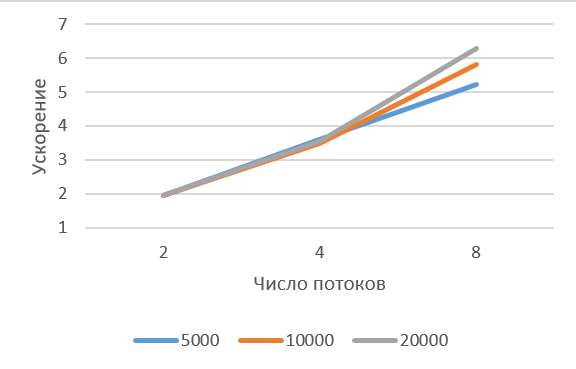
Из таблицы видно, что программа быстрее всего исполнялась на 8 потоках.

*Таблица 4 – Ускорение умножения матрицы на вектор, с*

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность | Число потоков | | |
| 2 | 4 | 8 |
| 5000 | 1,96 | 3,5982653 | 5,22 |
| 10000 | 1,96 | 3,4891498 | 5,82 |
| 20000 | 1,95 | 3,5503557 | 6,30 |

Из таблицы видно, что максимальное ускорение при 8 потоках, однако ускорение не достигает 8.

*График 2 – Ускорение умножения матрицы на вектор, с*



Листинг программы приведен в [приложении А задание 5](#_Задание_5.).

**Задание 6**

Написать программу, вычисляющую сумму чисел от 1 до N. Число N вводится как параметр командной строки. Замерить время работы программы. Результат вывести на консоль.

Распараллелить вычисление суммы с использованием std::thread. Результат, полученный каждым потоком, записать в разделяемую переменную. Для синхронизации потоков использовать std::mutex.

Замерить время работы программы для N = 107, 108 и 109 на 1, 2, 4 и 8 потоках. На каждом примере запустить не менее 3 раз. В таблицы занести среднее время выполнения на одном примере в секундах и ускорение. Построить диаграмму зависимости ускорения от числа потоков (три графика на одной диаграмме).

**Полученные результаты**

*Проверка работы алгоритма.*

*Таблица 5 – Время вычисление суммы, с*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размерность | Число потоков | | | |
| 1 | 2 | 4 | 8 |
| 107 | 0,0023571 | 0,0014051 | 0,001276 | 0,0011502 |
| 108 | 0,0144072 | 0,0077773 | 0,005149 | 0,004044 |
| 109 | 0,135902 | 0,0690642 | 0,05326 | 0,0318545 |

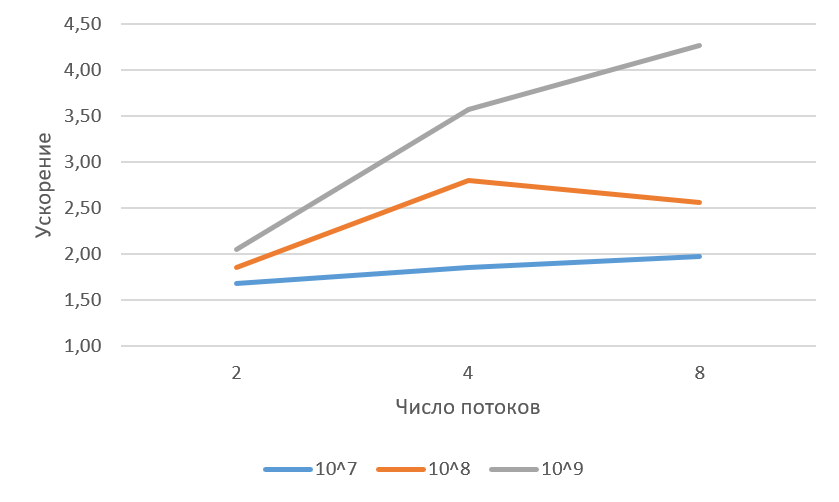
Из таблицы видно, что программа быстрее всего исполнялась на 8 потоках.

*Таблица 6 – Ускорение вычисления суммы, с*

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность | Число потоков | | |
| 2 | 4 | 8 |
| 107 | 1,68 | 1,8472571 | 2,05 |
| 108 | 1,85 | 2,7980035 | 3,56 |
| 109 | 1,97 | 2,551671 | 4,27 |

Из таблицы видно, что максимальное ускорение при 8 потоках, однако ускорение не достигает 8.

*График 3 – Ускорение вычисления суммы, с*



Листинг программы приведен в [приложении А задание 6](#_Задание_6.).

Вывод

В ходе лабораторной работы я получил навыки работы с потоками в языке C++. Были произведены замеры времени выполнения программ и на их основании сделаны выводы. Все тесты проводились на процессоре AMD Ryzen 5 3600 6/12 4.2ГГц.

# **Приложения**

## **Приложение А. Листинги программ**

### Задание 0.

#include "task0.h"  
  
void foo();  
  
int main()  
{  
 std::thread myTh(foo);  
  
 std::cout << "Main thread id = " << std::this\_thread::get\_id() << std::endl;  
  
 myTh.detach();  
  
 return 0;  
}  
  
void foo()  
{  
 std::cout << "Thread start..." << std::endl;  
 for (int i = 0; i < 10; i++) {  
 std::cout << "Thread id = " << std::this\_thread::get\_id() << std::endl;  
 std::this\_thread::sleep\_for(std::chrono::seconds(1));  
 }  
  
 std::cout << "Thread finish!" << std::endl;  
}

### Задание 1.

#include "task1.h"  
  
void foo(int);  
  
int main()  
{  
 std::thread myTh1(foo, 1);  
 std::thread myTh2(foo, 2);  
  
 std::cout << "Main thread id = " << std::this\_thread::get\_id() << std::endl;  
  
 myTh1.join();  
 myTh2.join();  
  
 return 0;  
}  
  
void foo(int n)  
{  
 std::cout << "Thread " << std::this\_thread::get\_id() << " start..." << std::endl;  
 for (int i = n; i <= 100; i += 2) {  
 std::cout << "Thread " << n << " Num = " << i << std::endl;  
 std::this\_thread::sleep\_for(std::chrono::seconds(1));  
 }  
  
 std::cout << "Thread " << std::this\_thread::get\_id() << " finish!" << std::endl;  
}

### Задание 2.

#include "task2.h"  
  
void foo();  
  
int main()  
{  
 srand(time(nullptr));  
  
 int P = 1;  
 std::cout << "Input P: ";  
 std::cin >> P;  
 if (P < 1 || P > 10) {  
 return -1;  
 }  
 std::cout << std::endl;  
  
 auto\* threads = new std::thread[P];  
  
 for (int i = 0; i < P; i++) {  
 threads[i] = std::thread(foo);  
 }  
  
 std::cout << "Main thread id = " << std::this\_thread::get\_id() << std::endl;  
  
 for (int i = 0; i < P; i++) {  
 threads[i].join();  
 }  
  
 delete[] threads;  
 return 0;  
}  
  
void foo()  
{  
 std::cout << "Thread " << std::this\_thread::get\_id() << " start..." << std::endl;  
 for (int i = 0; i < 100; i++) {  
 std::cout << "Thread id = " << std::this\_thread::get\_id() << std::endl;  
 std::this\_thread::sleep\_for(std::chrono::milliseconds(1000 + (rand() % (2000 - 1000 + 1))));  
 }  
  
  
 std::cout << "Thread " << std::this\_thread::get\_id() << " finish!" << std::endl;  
}

### Задание 3.

#include "task3.h"  
  
int cnt = 0;  
void foo(int, std::mutex&);  
  
int main()  
{  
 srand(time(nullptr));  
  
 std::cout << "Cnt = " << cnt << std::endl;  
  
 int P = 1;  
 std::cout << "Input P: ";  
 std::cin >> P;  
 if (P < 1 || P > 10) {  
 return -1;  
 }  
 std::cout << std::endl;  
  
 auto\* threads = new std::thread[P];  
 std::mutex mx;  
  
 for (int i = 0; i < P; i++) {  
 threads[i] = std::thread(foo, i + 1, ref(mx));  
 }  
  
 std::unique\_lock<std::mutex> ulmx(mx);  
 std::cout << "Main thread id = " << std::this\_thread::get\_id() << std::endl;  
 ulmx.unlock();  
  
 for (int i = 0; i < P; i++) {  
 threads[i].join();  
 }  
  
 std::cout << "Cnt = " << cnt << std::endl;  
  
 delete[] threads;  
 return 0;  
}  
  
void foo(int id, std::mutex& mx)  
{  
 std::unique\_lock<std::mutex> ulmx(mx);  
 std::cout << "Thread " << std::this\_thread::get\_id() << " start..." << std::endl;  
 ulmx.unlock();  
  
 while(cnt < 100) {  
 std::this\_thread::sleep\_for(std::chrono::milliseconds(10));  
 std::lock\_guard<std::mutex> lgmx(mx);  
 if (cnt < 100) {  
 cnt += id;  
 std::cout << "Cnt for thread " << std::this\_thread::get\_id() << " = " << cnt << std::endl;  
 }  
 }  
  
 ulmx.lock();  
 std::cout << "Thread " << std::this\_thread::get\_id() << " finish!" << std::endl;  
 ulmx.unlock();  
}

### Задание 4.

#include "task4.h"  
  
int\* arr;  
int N;  
int\* create\_random\_vector(int);  
void print\_vector(int\*);  
void foo(int, int, std::mutex&);  
int largest\_prime\_divisor(int);  
  
int main()  
{  
 srand(time(nullptr));  
  
 int P = 1;  
 std::cout << "Input P: ";  
 std::cin >> P;  
 if (P < 1 || P > 10) {  
 return -1;  
 }  
 std::cout << std::endl;  
  
 auto\* threads = new std::thread[P];  
 std::mutex mx;  
  
 std::cout << "Input N: ";  
 std::cin >> N;  
  
 arr = create\_random\_vector(N);  
 print\_vector(arr);  
  
 int k = N / P;  
  
 std::chrono::time\_point<std::chrono::high\_resolution\_clock> start, end;  
 start = std::chrono::high\_resolution\_clock::now();  
  
 for (int i = 0; i < P; i++) {  
 threads[i] = std::thread(foo, k \* i, k \* (i + 1) - 1, ref(mx));  
 }  
  
 for (int i = 0; i < P; i++) {  
 threads[i].join();  
 }  
  
 end = std::chrono::high\_resolution\_clock::now();  
 std::chrono::duration<double> diff = end - start;  
  
 double time = diff.count();  
 std::cout << "Time to process: " << time << "s" << std::endl;  
  
 print\_vector(arr);  
  
 delete[] arr;  
 delete[] threads;  
 return 0;  
}  
  
int\* create\_random\_vector(int size)  
{  
 int \*result = new int[size];  
 for (int i = 0; i < size; i++) {  
 *//result[i] = 100000 + (rand() % (1000000 - 100000 + 1));* result[i] = 100000 + (rand() % (100 - 3 + 1));  
 }  
 return result;  
}  
  
void print\_vector(int\* array)  
{  
 for (int i = 0; i < N; i++) {  
 std::cout << array[i] << " ";  
 }  
 std::cout << std::endl;  
}  
  
void foo(int from, int to, std::mutex& mx)  
{  
 for (int i = from; i <= to; i++) {  
 *//std::this\_thread::sleep\_for(std::chrono::milliseconds(10));* arr[i] = largest\_prime\_divisor(arr[i]);  
 *//std::lock\_guard<std::mutex> lgmx(mx);  
 //std::cout << "Thread " << std::this\_thread::get\_id() << " arr[" << i << "] = " << arr[i] << std::endl;* }  
}  
  
int largest\_prime\_divisor(int n)  
{  
 int maxPrime = -1;  
  
 *// Пока число четное, делим его на 2* while (n % 2 == 0) {  
 maxPrime = 2;  
 n /= 2;  
 }  
  
 *// Пока число делится на 3, делим его на 3* while (n % 3 == 0) {  
 maxPrime = 3;  
 n /= 3;  
 }  
  
 *// Проходим по натуральным числам, которые не делятся на 2 и 3* for (int i = 5; i <= sqrt(n); i += 6) {  
 *// Пока число делится на i, делим его на i* while (n % i == 0) {  
 maxPrime = i;  
 n /= i;  
 }  
 *// Пока число делится на i + 2, делим его на i + 2* while (n % (i + 2) == 0) {  
 maxPrime = i + 2;  
 n /= i + 2;  
 }  
 }  
  
 *// n - простое число больше 4* if (n > 4)  
 maxPrime = n;  
  
 return maxPrime;  
}

### Задание 5.

#include "task5.h"  
  
int N;  
std::vector<std::vector<int>> matrix;  
std::vector<int> vector;  
std::vector<int> result;  
  
std::vector<int> create\_random\_vector(int);  
void print\_vector(std::vector<int>);  
std::vector<std::vector<int>> create\_random\_matrix(int);  
void print\_matrix(std::vector<std::vector<int>>);  
std::vector<int> multiply\_sequentially(std::vector<std::vector<int>>, std::vector<int>);  
void foo(int, int, std::mutex&);  
  
int main()  
{  
 srand(time(nullptr));  
  
 int P = 1;  
 std::cout << "Input P: ";  
 std::cin >> P;  
 if (P < 1 || P > 10) {  
 return -1;  
 }  
 std::cout << std::endl;  
  
 auto\* threads = new std::thread[P];  
 std::mutex mx;  
  
 std::cout << "Input N: ";  
 std::cin >> N;  
  
 matrix = create\_random\_matrix(N);  
 *//print\_matrix(matrix);* vector = create\_random\_vector(N);  
 *//print\_vector(vector);  
 //std::cout << std::endl;* std::chrono::time\_point<std::chrono::high\_resolution\_clock> start, end;  
 start = std::chrono::high\_resolution\_clock::now();  
  
 std::vector<int> res = multiply\_sequentially(matrix, vector);  
  
 end = std::chrono::high\_resolution\_clock::now();  
 std::chrono::duration<double> diff = end - start;  
  
 *//print\_vector(res);* double time = diff.count();  
 std::cout << "Time to process sequentially: " << time << "s" << std::endl;  
  
 for (int i = 0; i < N; ++i) {  
 result.push\_back(0);  
 }  
  
 int k = N / P;  
  
 start = std::chrono::high\_resolution\_clock::now();  
  
 for (int i = 0; i < P; i++) {  
 threads[i] = std::thread(foo, k \* i, k \* (i + 1) - 1, ref(mx));  
 }  
  
 for (int i = 0; i < P; i++) {  
 threads[i].join();  
 }  
  
 end = std::chrono::high\_resolution\_clock::now();  
 diff = end - start;  
  
 *//print\_vector(result);* time = diff.count();  
 std::cout << "Time to process parallel: " << time << "s" << std::endl;  
  
 delete[] threads;  
 return 0;  
}  
  
std::vector<int> create\_random\_vector(int size)  
{  
 std::vector<int> res(size);  
 for (int i = 0; i < size; i++) {  
 res[i] = (rand() % 3 + 1);  
 }  
 return res;  
}  
  
void print\_vector(std::vector<int> v)  
{  
 for (int i : v) {  
 std::cout << i << " ";  
 }  
 std::cout << std::endl;  
}  
  
std::vector<std::vector<int>> create\_random\_matrix(int size)  
{  
 std::vector<std::vector<int>> res;  
 for (int i = 0; i < size; i++) {  
 res.emplace\_back();  
 for (int j = 0; j < size; ++j) {  
 res[i].push\_back(rand() % 3 + 1);  
 }  
 }  
 return res;  
}  
  
void print\_matrix(std::vector<std::vector<int>> m)  
{  
 for (const std::vector<int>& line : m) {  
 for (int item : line) {  
 std::cout << item << " ";  
 }  
 std::cout << std::endl;  
 }  
 std::cout << std::endl;  
}  
  
std::vector<int> multiply\_sequentially(std::vector<std::vector<int>> m, std::vector<int> v) {  
 std::vector<int> res;  
  
 for (int i = 0; i < m.size(); i++) {  
 res.push\_back(0);  
 for (int j = 0; j < v.size(); j++) {  
 res[i] += m[i][j] \* v[j];  
 }  
 }  
   
 return res;  
}  
  
void foo(int from, int to, std::mutex& mx)  
{  
 for (int i = from; i <= to; i++) {  
 for (int j = 0; j < vector.size(); j++) {  
 result[i] += matrix[i][j] \* vector[j];  
 }  
 }  
}

### Задание 6.

#include "task6.h"  
  
int N;  
std::vector<long long> sums;  
  
void foo(int, int, int, std::mutex&);  
  
int main(int argc, char \*argv[])  
{  
 if (argc != 2) {  
 return -1;  
 }  
  
 N = atoi(argv[1]);  
  
 srand(time(nullptr));  
  
 int P = 1;  
 std::cout << "Input P: ";  
 std::cin >> P;  
 if (P < 1 || P > 10) {  
 return -1;  
 }  
 std::cout << std::endl;  
  
 auto\* threads = new std::thread[P];  
 std::mutex mx;  
  
 int k = N / P;  
 for (int i = 0; i < P; ++i) {  
 sums.push\_back(0);  
 }  
  
 std::chrono::time\_point<std::chrono::high\_resolution\_clock> start, end;  
 start = std::chrono::high\_resolution\_clock::now();  
  
 for (int i = 0; i < P; i++) {  
 int from = k \* i + 1;  
 int to = i == P - 1 ? k \* (i + 1) + N % P : k \* (i + 1);  
 threads[i] = std::thread(foo, i, from, to, ref(mx));  
 }  
  
 for (int i = 0; i < P; i++) {  
 threads[i].join();  
 }  
  
 long long result = 0;  
 for (int i = 0; i < P; i++) {  
 result += sums[i];  
 }  
  
 end = std::chrono::high\_resolution\_clock::now();  
 std::chrono::duration<double> diff = end - start;  
  
 std::cout << "Result " << result << std::endl;  
  
 double time = diff.count();  
 std::cout << "Time to sum: " << time << "s" << std::endl;  
  
 delete[] threads;  
 return 0;  
}  
  
void foo(int l, int from, int to, std::mutex& mx)  
{  
 long long sum = 0;  
 for (int i = from; i <= to; i++) {  
 sum += i;  
 }  
  
 sums[l] = sum;  
 */\*std::unique\_lock<std::mutex> ulmx(mx);  
 std::cout << "Thread " << l << " from = " << from << " to = " << to << " sums[" << l << "] = " << sums[l] << std::endl;  
 ulmx.unlock();\*/*}