ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

ОТЧЕТ ПО ДОМАШНЕЙ РАБОТЕ №4

Исполнитель
студент группы **БПИ196**
Шилова Ксения Алексеевна

ВАРИАНТ 29

Формулировка задачи:

Вычислить интеграл:
$\int_a^b f(x)dx$,
используя метод прямоугольников. Входные данные: вещественные числа **a** и **b**, функция **f(x)** задается с использованием описания в программе в виде отдельной функции. При суммировании использовать принцип дихотомии.
Протестировать на различных функциях.

28 ноября 2020 г.

Описание работы программы

Для того, чтобы продемонстрировать работу OPEN mp при реализации данной задачи, я написала **рекурсивную** программу для численного вычисления интеграла и **итеративную**, но с имитацией дихотомии (разбила на 8 равных частей весь отрезок).

Общее для обеих программ (работа с пользователем)

1. Считывание интервала интегрирования с консоли:
   readInterval()
2. Отображение функций, возможных для вычисления, в консоль
   displayFunctions()
   Возможные функции:

```
Type number from 1 to 6 to choose the function:
f1(x) = 1
f2(x) = x
f3(x) = pow(x,3)
f4(x) = sin(x)
f5(x) = exp(x)
f6(x) = sqrt(x)
Type "exit" to exit the program.
```

3. Алгоритм считывания нужного номера функции, и вычисление интеграла до тех пор, пока пользователь не введет **exit**:

```
//ЦИКЛ ПОВТОРА ВЫБОРА ТЕСТИРУЕМОЙ ФУНКЦИИ (ИЗ 6-ТИ ПРЕДЛОЖЕННЫХ)
do {
    bool again = false;

    displayFunctions(); //отображаем возможные для выбора функции

    //СЧИТЫВАНИЕ НОМЕРА ФУНКЦИИ ИЛИ ДРУГОЙ ИНФОРМАЦИИ
    std::string s;
    double (*function)(double) = nullptr;
    std::cin >> s;
    //если это номер, то берем нужную функцию
    if (s == "1") function = f1;
    else if (s == "2") function = f2;
    else if (s == "3") function = f3;
    else if (s == "4") function = f4;
    else if (s == "5") function = f5;
    else if (s == "6") { //нельзя, если границы отрицательные
        if (a < 0) {
            std::cout << "Function f6 is not defined in negative numbers. " << std::endl;
            again = true;
        }
        function = f6;
    }
    //если выход
    else if (s == "exit") break;
    //иначе - некорректная строка
    else {
        std::cout << "Incorrect number of function. Type number (1, ..., 6)  or \"exit\"" << std::endl;
        again = true;
    }
    //-----------------------------------------------------
```

Алгоритм вычисления интеграла с помощью рекурсии

Для того чтобы вычислить интеграл с помощью рекурсии, необходимо задать некий небольшой отрезок (eps), чтобы выходить из функции, когда разность между правой и левой границей становится меньше, чем этот отрезок.

Если условия выхода не выполняется, то функция должна вернуть сумму интегралов на двух равных отрезках, на которые нужно разбить основной отрезок.

Для того, чтобы распараллелить, используем #pragma sections.

Ставим блоки section, так как нам не важен порядок подсчета на каждом из отрезков, на которые мы делим текущий. То есть, все эти блоки независимы, поэтому можно использовать такой способ. Кроме того, подсчет интеграла осуществляется в переменную sum, которая общая для всех потоков.

```
double eps = 0.001;
double sum = 0;
double rectIntegral(
    const function<double(double)>& fun,
    const double a, const double b)
{
    if (b - a < eps)
    {
        return fun((a + b) / 2) * (b - a);
    }

    double sum1, sum2;
#pragma omp parallel
    {
#pragma sections
        {
        #pragma section
        sum1 = rectIntegral(fun, a, (a + b) / 2);
        #pragma section
        sum2 = rectIntegral(fun, (a + b) / 2, b);
        }
    }
    return sum1 + sum2;
```

Я также нашла и другой способ для того, чтобы распараллелить рекурсивную функцию – использование #pragma omp task.

```
double eps = 0.001;
double sum = 0;
double rectIntegral(
    const function<double(double)>& fun,
    const double a, const double b)
{
    if (b - a < eps)
    {
        return fun((a + b) / 2) * (b - a);
    }

    double sum1, sum2, result;
#pragma omp parallel
    {
        #pragma omp task shared(sum1)
        sum1 = rectIntegral(fun, a, (a + b) / 2);
        #pragma omp task shared(sum2)
        sum2 = rectIntegral(fun, (a + b) / 2, b);
        #pragma omp taskwait
        result = sum1 + sum2;
    }
    return result;
```

Но, к сожалению, Visual Studio не поддерживает OPEN MP 3.0, а #pragma omp task есть только в OPEN MP 3.0. Поэтому я не смогла протестировать это на своем компьютере, но оставила закомментированный кусок кода с этим способом в программе.

Алгоритм вычисления интеграла итеративно

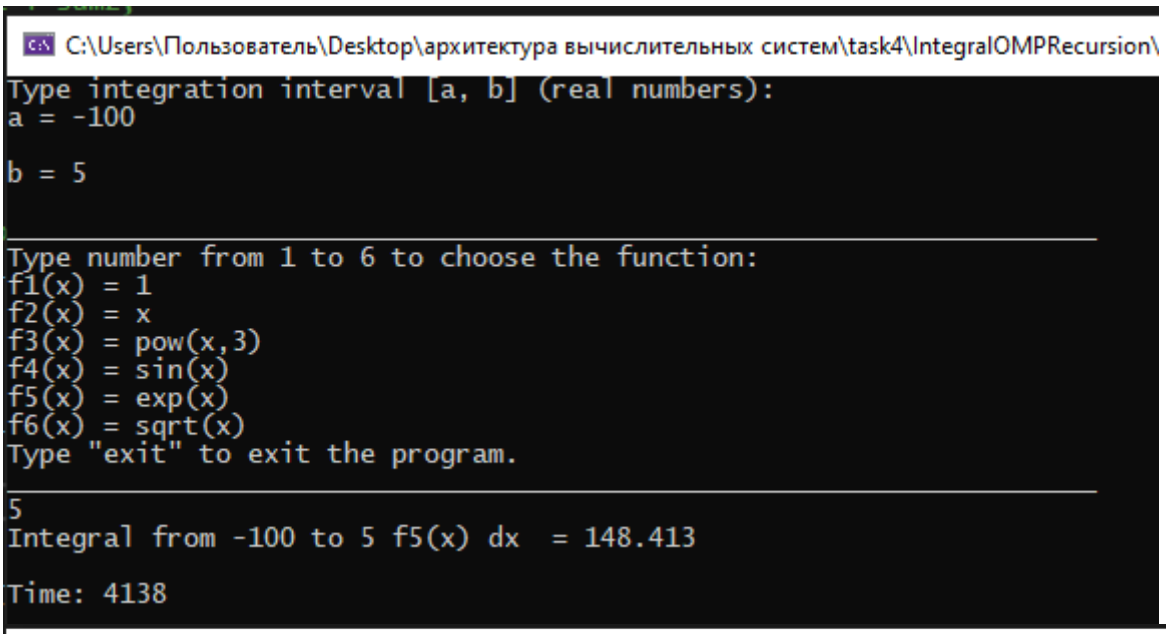
Для того, чтобы посчитать интеграл итеративно, разобьем его на 8 частей (это похоже на то, что мы пользуемся методом дихотомии – разбиваем на 8 частей весь отрезок для каждого потока). Таким образом, каждый поток посчитает свой отрезок. Внутри каждого отрезка, также пользуемся итеративным способом вычисления (разбиваем на более мелкие отрезки) и используем для внутреннего цикла #pragma omp for nowait, чтобы распараллелить еще и внутренний цикл. То есть nowait можно было бы и не использовать, так как нагрузка для каждого ядра все равно была бы равномерной, но для логики и универсальности метода я использую nowait.

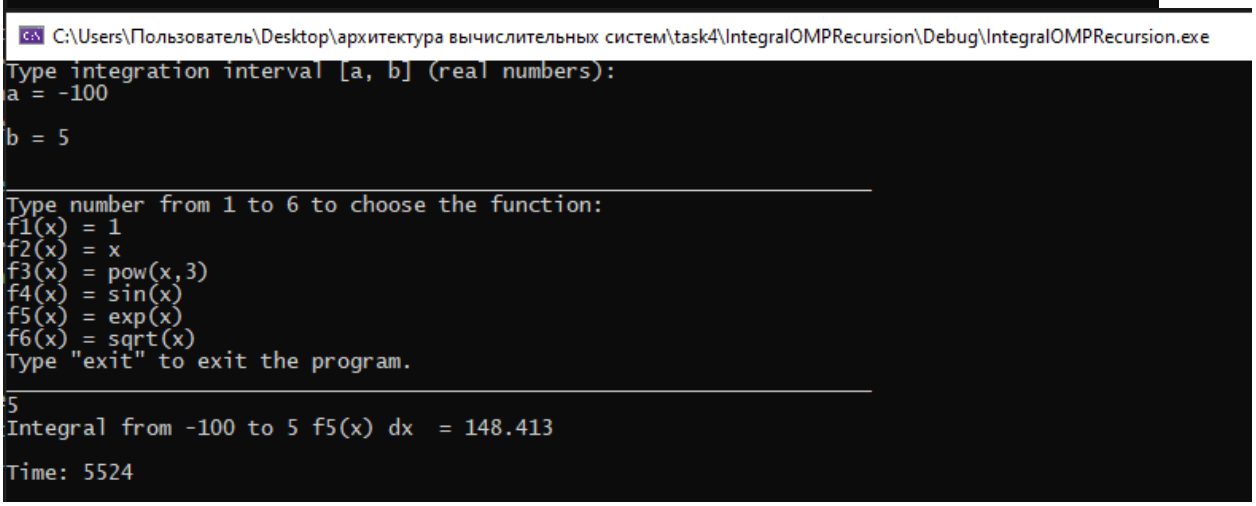
```
double rectIntegral(
    const function<double(double)>& fun,
    const double a, const double b, const int n)
{
    //разбиваем на 8 отрезков, подобны методу дихотомии
    double oneInterval = (b - a) / amountOfThreads;
    double sum = 0;

#pragma omp parallel num_threads(8) reduction (+: sum)
    {
        //проходимся по всем потокам
        for (int counter = 0; counter < amountOfThreads; counter++) {

            double delta = oneInterval / n;
#pragma omp for nowait
            //внутри одной из восьми частей производим сложение площадей прямоугольников
            for (int i = 0; i < n; i++) {
                //counter = 0 =>   fun(a)...fun(a + oneInterval)
                //counter = 1 =>   fun(a + oneInterval)...fun(a + 2*oneInterval)
                //counter = 7 =>   fun(a + 7*oneInterval)...fun(b)
                sum += fun(a + counter*oneInterval + i*delta) * delta;
            }
        }
#pragma omp critical
        {
            std::cout << "Thread number " << omp_get_thread_num() << ": integral = " << sum << endl;
        }
    }  // pragma omp parallel
    return sum;
}
```

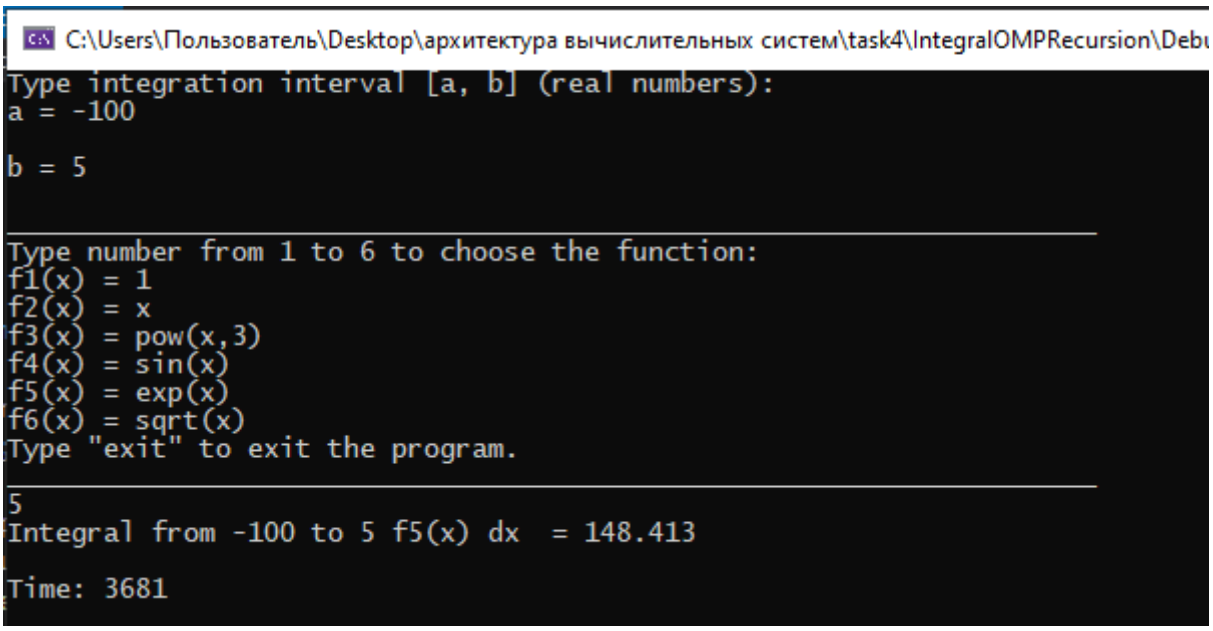
Тестирование

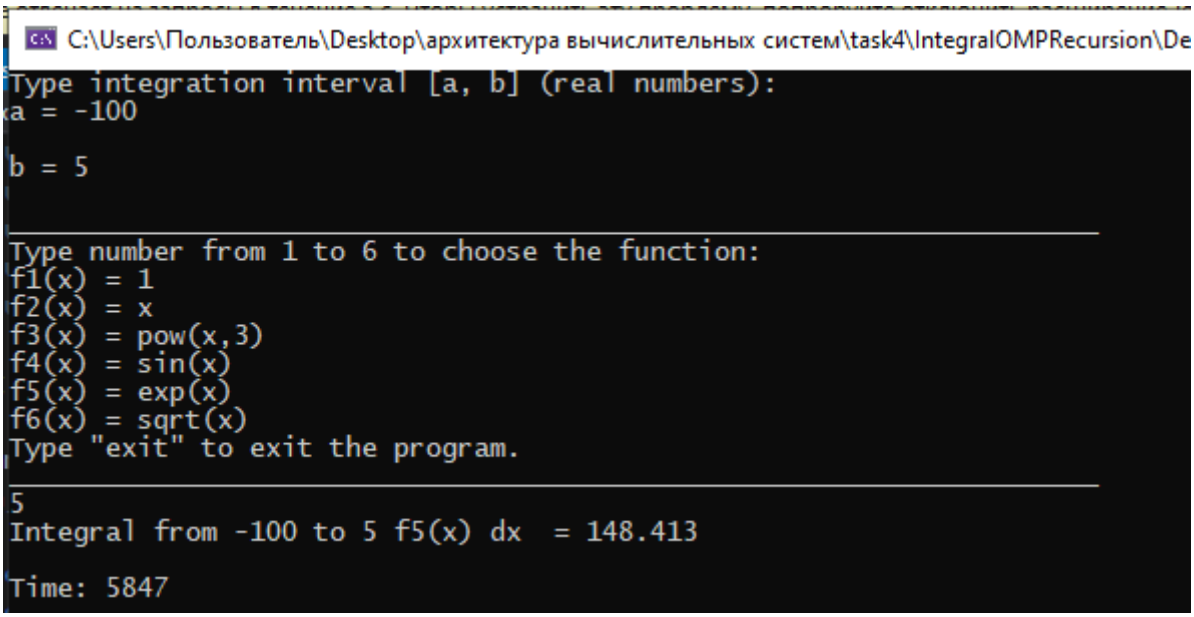
1. Протестируем подсчет интеграла по одной из предложенных функций. Например, проинтегрируем экспоненциальную функцию на интервале -100, 5. Запустим несколько раз, чтобы усреднить время

   1.1 Рекурсивный параллелизм


C:\Users\Пользователь\Desktop\архитектура вычислительных систем\task4\IntegralOMPRecursion\
Type integration interval [a, b] (real numbers):
a = -100
b = 5
Type number from 1 to 6 to choose the function:
f1(x) = 1
f2(x) = x
f3(x) = pow(x,3)
f4(x) = sin(x)
f5(x) = exp(x)
f6(x) = sqrt(x)
Type "exit" to exit the program.
5
Integral from -100 to 5 f5(x) dx = 148.413
Time: 4138

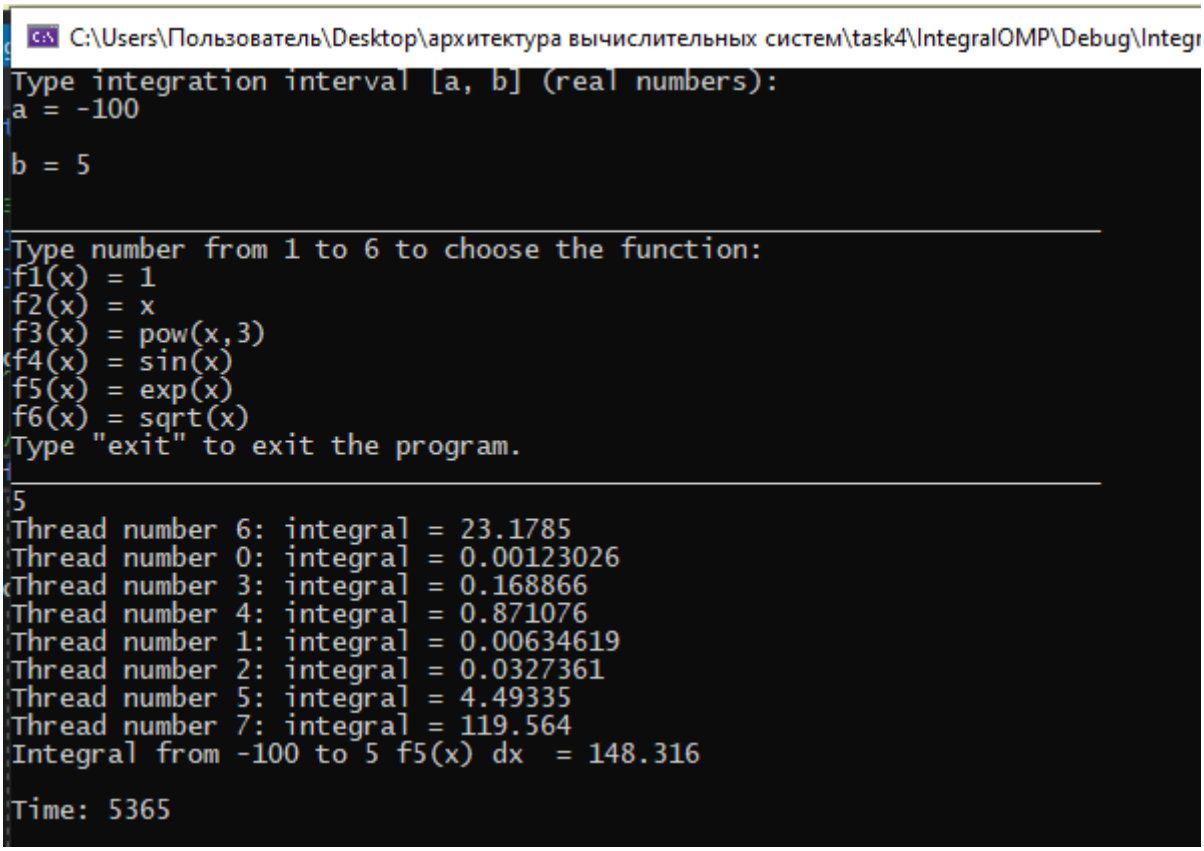

C:\Users\Пользователь\Desktop\архитектура вычислительных систем\task4\IntegralOMPRecursion\Debug\IntegralOMPRecursion.exe
Type integration interval [a, b] (real numbers):
a = -100
b = 5
Type number from 1 to 6 to choose the function:
f1(x) = 1
f2(x) = x
f3(x) = pow(x,3)
f4(x) = sin(x)
f5(x) = exp(x)
f6(x) = sqrt(x)
Type "exit" to exit the program.
5
Integral from -100 to 5 f5(x) dx = 148.413
Time: 5524

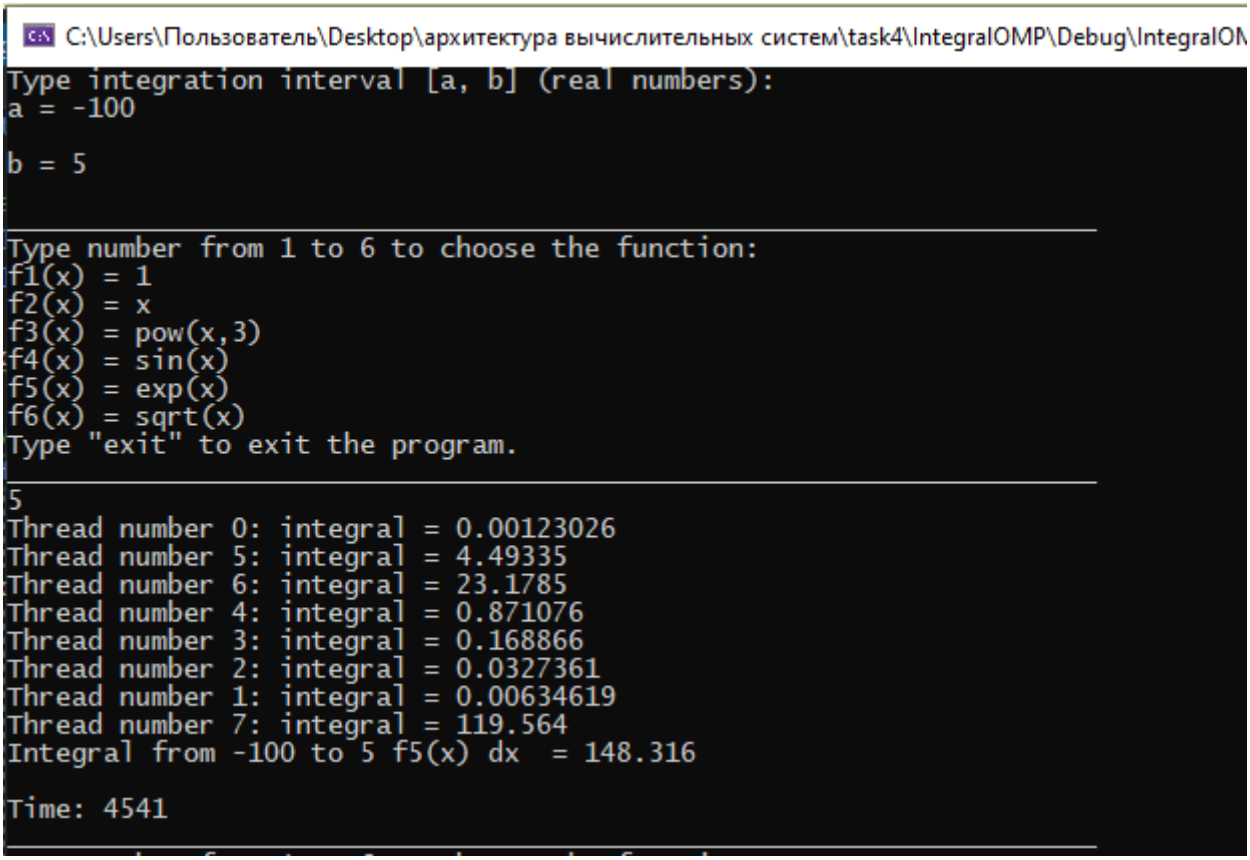

C:\Users\Пользователь\Desktop\архитектура вычислительных систем\task4\IntegralOMPRecursion\Debu
Type integration interval [a, b] (real numbers):
a = -100
b = 5
Type number from 1 to 6 to choose the function:
f1(x) = 1
f2(x) = x
f3(x) = pow(x,3)
f4(x) = sin(x)
f5(x) = exp(x)
f6(x) = sqrt(x)
Type "exit" to exit the program.
5
Integral from -100 to 5 f5(x) dx = 148.413
Time: 3681

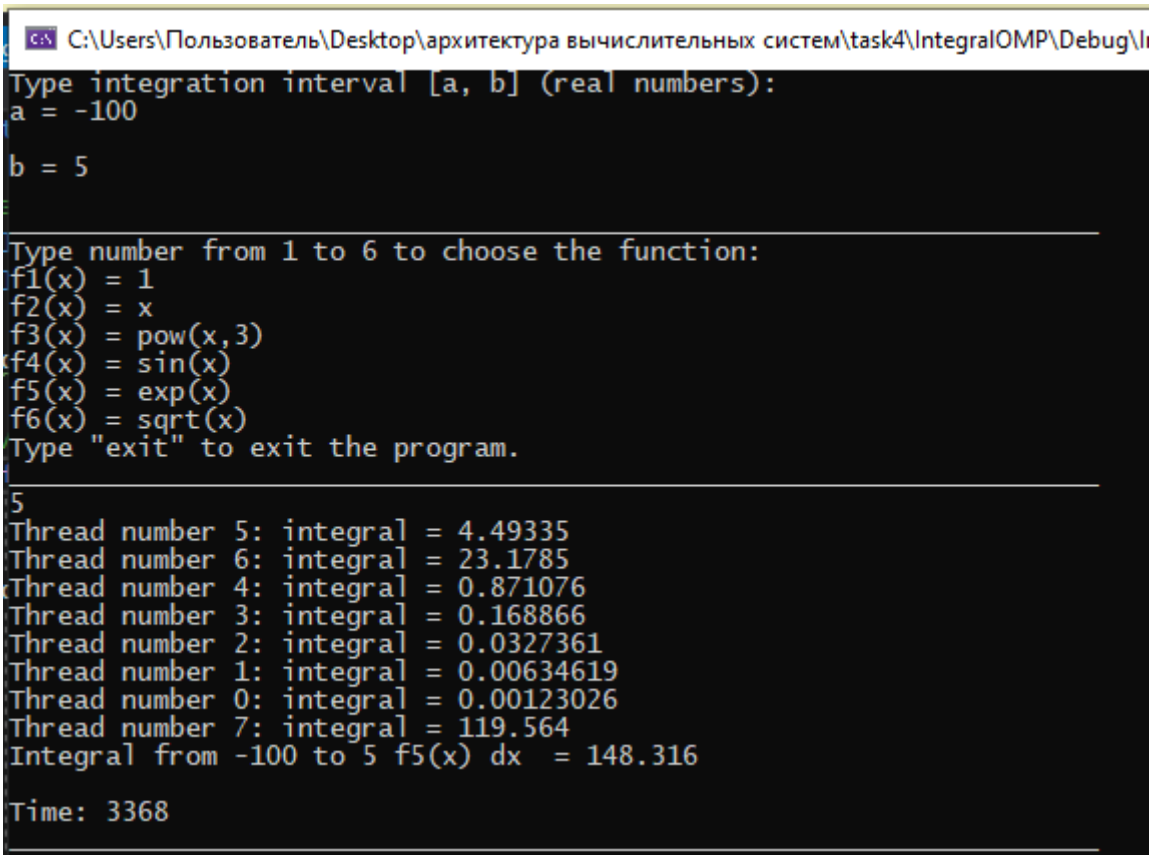


Итоговое время: около **4500** мс.

1.2 Итеративно

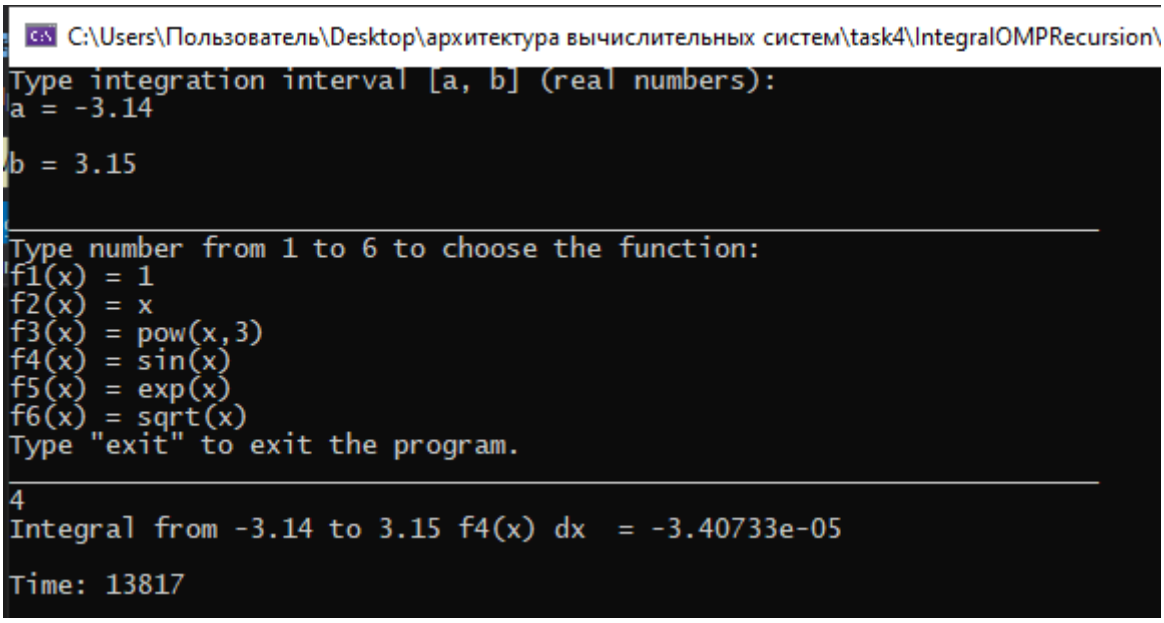


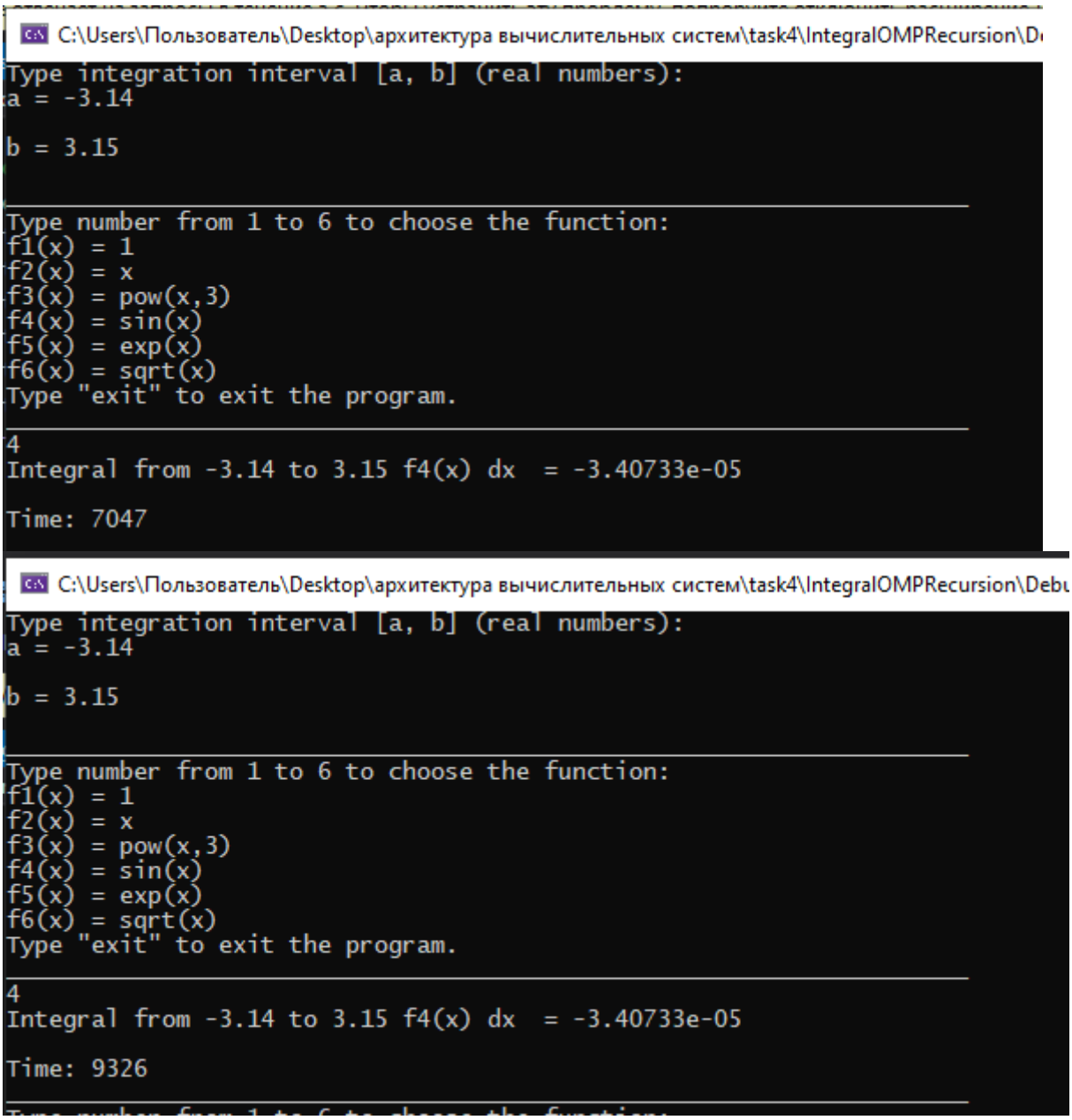




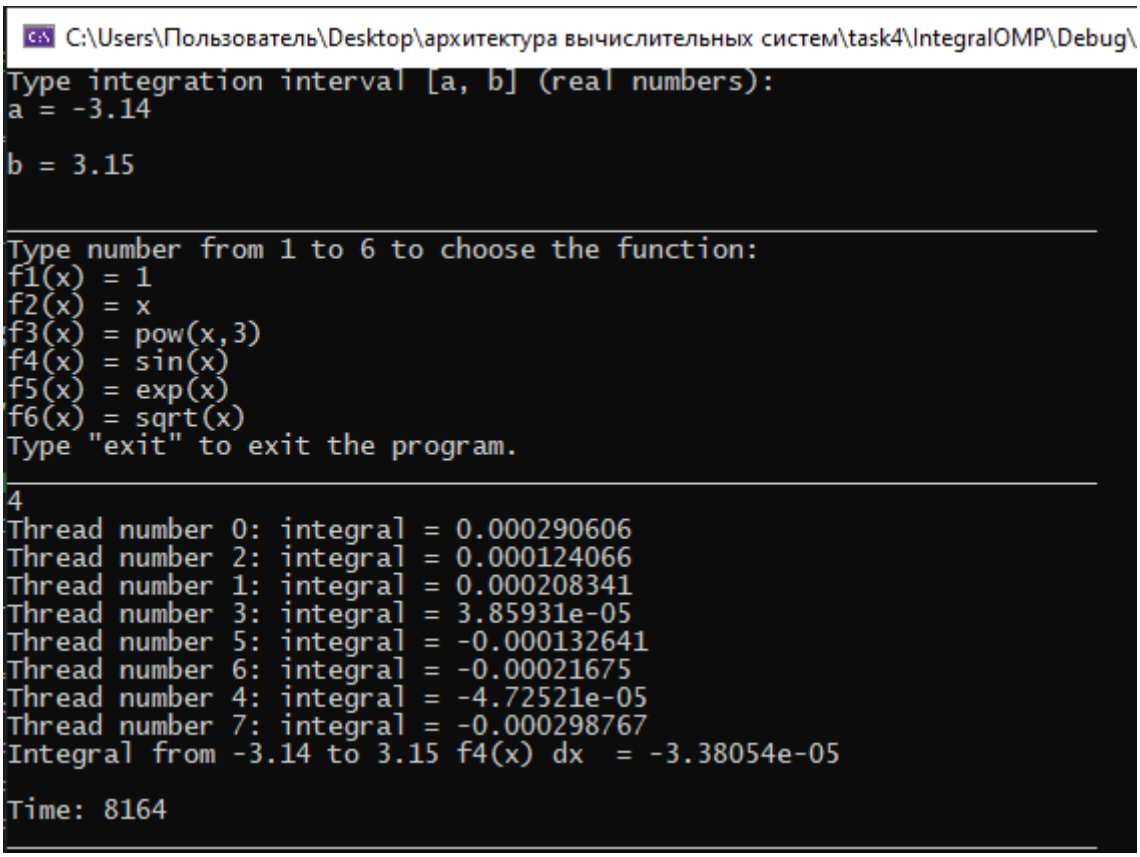
Итоговое время: около **4500** мс.

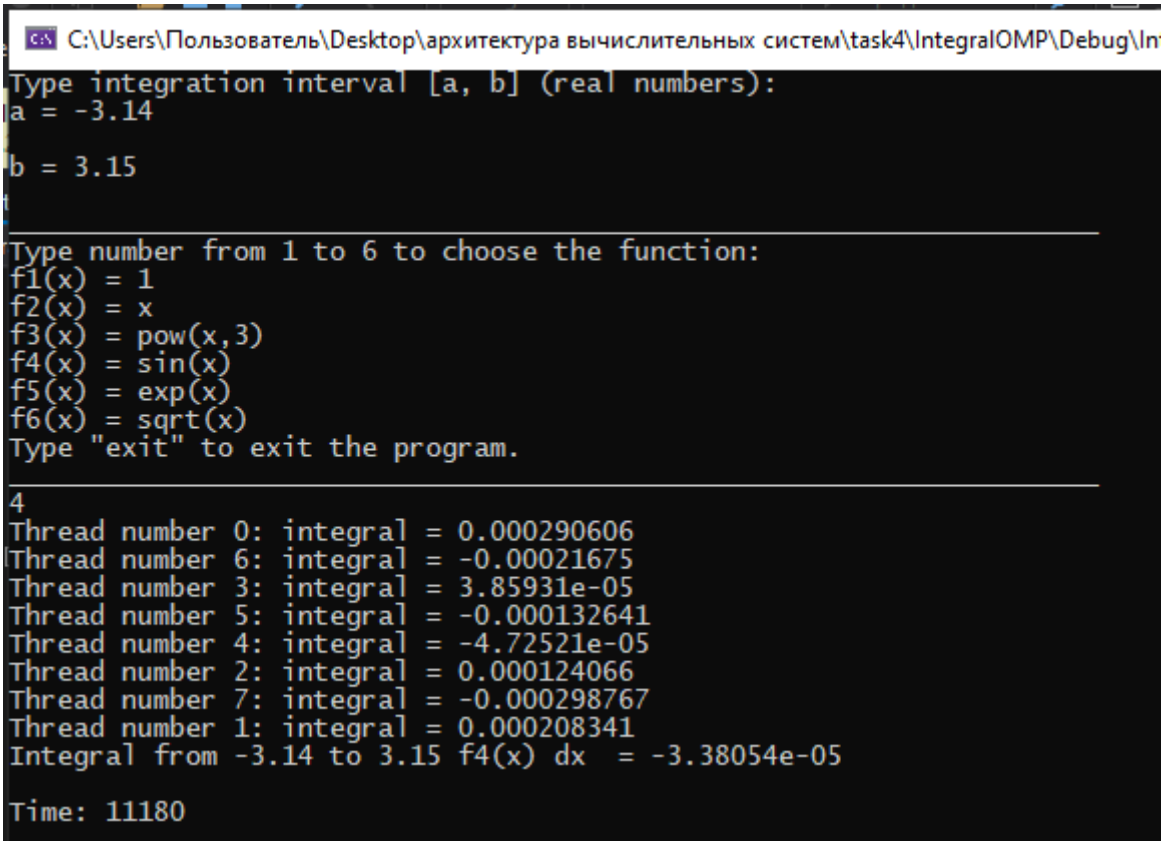
2. Функция f(x) = sin(x)
   2.1 Рекурсивная функция

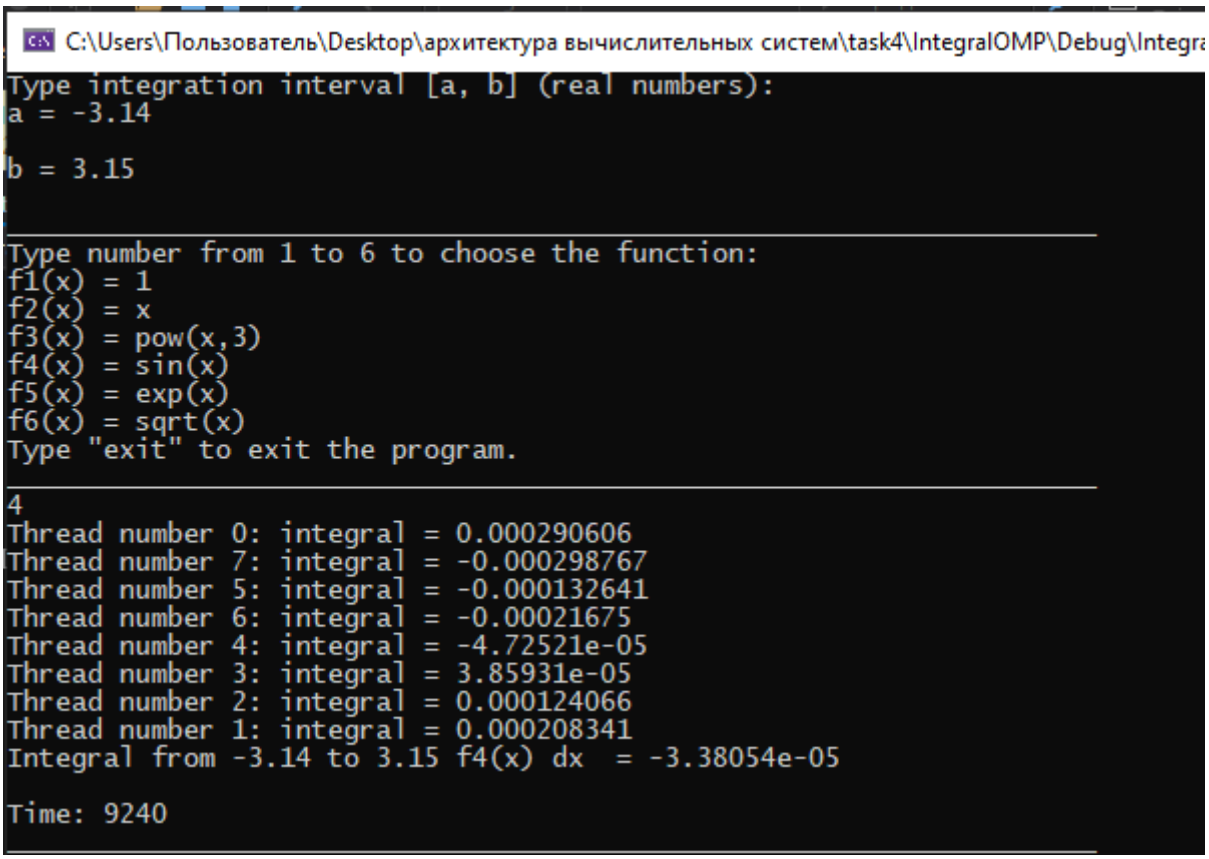




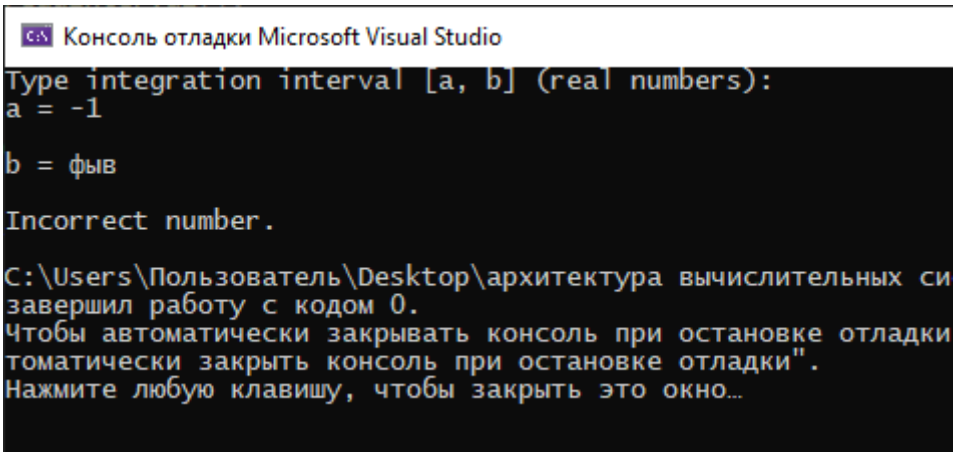
2.2 Итеративный способ

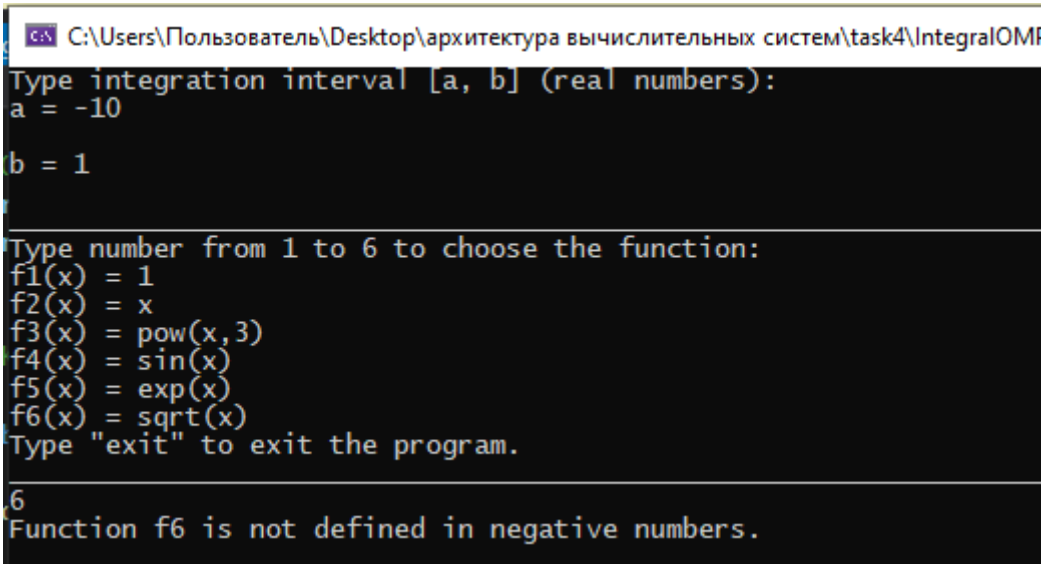

C:\Users\Пользователь\Desktop\архитектура вычислительных систем\task4\IntegralOMP\Debug\
Type integration interval [a, b] (real numbers):
a = -3.14
b = 3.15
Type number from 1 to 6 to choose the function:
f1(x) = 1
f2(x) = x
f3(x) = pow(x,3)
f4(x) = sin(x)
f5(x) = exp(x)
f6(x) = sqrt(x)
Type "exit" to exit the program.
4
Thread number 0: integral = 0.000290606
Thread number 2: integral = 0.000124066
Thread number 1: integral = 0.000208341
Thread number 3: integral = 3.85931e-05
Thread number 5: integral = -0.000132641
Thread number 6: integral = -0.00021675
Thread number 4: integral = -4.72521e-05
Thread number 7: integral = -0.000298767
Integral from -3.14 to 3.15 f4(x) dx = -3.38054e-05
Time: 8164


C:\Users\Пользователь\Desktop\архитектура вычислительных систем\task4\IntegralOMP\Debug\In
Type integration interval [a, b] (real numbers):
a = -3.14
b = 3.15
Type number from 1 to 6 to choose the function:
f1(x) = 1
f2(x) = x
f3(x) = pow(x,3)
f4(x) = sin(x)
f5(x) = exp(x)
f6(x) = sqrt(x)
Type "exit" to exit the program.
4
Thread number 0: integral = 0.000290606
Thread number 6: integral = -0.00021675
Thread number 3: integral = 3.85931e-05
Thread number 5: integral = -0.000132641
Thread number 4: integral = -4.72521e-05
Thread number 2: integral = 0.000124066
Thread number 7: integral = -0.000298767
Thread number 1: integral = 0.000208341
Integral from -3.14 to 3.15 f4(x) dx = -3.38054e-05
Time: 11180

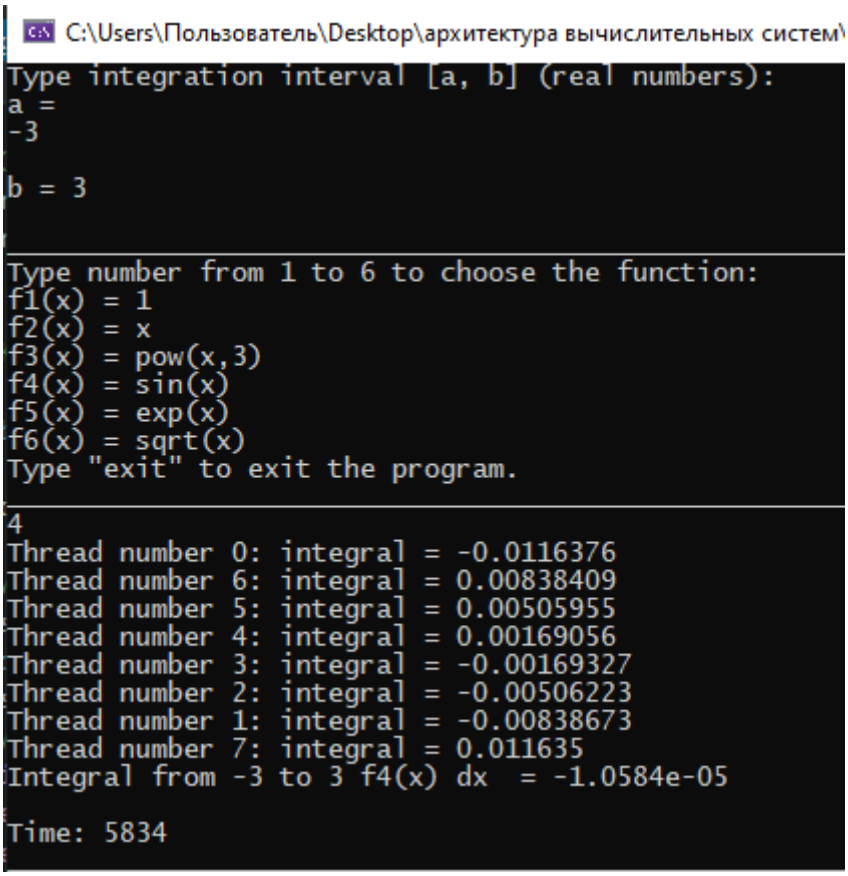


3. Проверка некорректного ввода





4. Проверка нечетной функции на симметричном интервале



Выводы

Время работы программ примерно одинаковое, =>так как мы наглядно видим, что удалось распараллелить программу итеративную, то можем сделать вывод, что рекурсию тоже удалось распараллелить.

Список литературы

1. https://pro-prof.com/archives/4335
2. http://mindhalls.ru/pragma-omp-directives-samples/
3. https://habr.com/ru/company/intel/blog/85273/