УДК 004.432.2

Арташкин Е.П.

студент

4 курс, специальность «Информатика и вычислительная техника»

Институт электроники и светотехники

Россия, г. Саранск

ОПТИМИЗАЦИЯ КОДА ЧЕРЕЗ РУЧНОЙ ТАЙМИНГ

Аннотация: *В данной статье рассматривается оптимизация кода путём ручного тайминга.*

Ключевые слова: *C++, тайминг, алгоритм, компьютер..*

Annotation: *This article discusses how to optimize the code by hand timing.*

Key words: *C++, timing, algorithm, computer.*

Иногда требуется сравнить скорость выполнения нескольких алгоритмов. Часто для этой цели используют профилировщики кода. Но бывает, что профилировщика нет под рукой, или работа с ним сложна, а нужно сравнить скорость работы фрагмента кода А со скоростью работы фрагмента кода Б в небольшой тестовой программе.

Шаблон программы

```
#include <iostream>
#include <ctime>
int main() {
    clock_t the_time;
    double elapsed_time;
    the_time = clock();
    // таймируемый код
    elapsed_time = double(clock() - the_time) / CLOCKS_PER_SEC;
    std::cout << "Elapsed time " << elapsed_time << " sec." << std::endl;
    return 0;
```

}

Эта конструкция позволяет узнать время выполнения фрагмента кода.

Включение заголовка <iostream> необходимо для вывода сообщений в стандартный вывод (т.е. на экран).

Включение заголовка <ctime> дает возможность пользоваться функцией clock() и сопутствующими типами и константами.

Единственная функция, которую мы будем использовать для таймирования, имеет следующую сигнатуру:

clock_t clock(void);

Функция возвращает время, прошедшее с момента запуска процесса (т.е. собственно с момента запуска данного приложения) в тиках таймера. Тип возвращаемого значения — clock_t обычно является псевдонимом типа long (определяется в <ctime>). Если функция не может получить системы время с начала процесса, функция возвращает значение -1, приведенное к типу clock_t.

Для перевода тиков таймера в секунды служит константа CLOCKS_PER_SEC, также определенная в<ctime>.

Алгоритм работы прост: перед выполнением таймируемого кода засекаем время, после выполнения таймируемого кода еще раз засекаем время, их разность даст нам искомое время, за которое выполнялся код. Остается только перевести его из тиков в секунды.

В приведенном шаблоне программы хотелось бы обратить внимание именно на последний шаг. Поскольку clock_t — целочисленный тип, а CLOCKS_PER_SEC — целочисленная константа, то для получения дробных частей секунд необходимо хотя бы одну из частей выражения привести к типу с плавающей точкой.

Несколько советов по выполнению таймирования

Между первым и вторым вызовами clock() не должно быть:

1) Операций ввода-вывода. Если, конечно, вы не хотите протестировать быстродействие вашего винчестера или подсистемы консольного вывода.

2) Операций запроса данных у пользователя (например, с клавиатуры).

Может оказаться, что используемый в вашем компьютере процессор слишком быстр для таймирования небольшого фрагмента кода. В этом случае разность значений между вызовами clock() будет равна 0.

Выход, как обычно, прост: заключить таймируемый фрагмент в цикл и крутить его там 1'000 или 1'000'000 раз. При этом накладывается дополнительное ограничение:

3) Таймируемый код не должен иметь побочных эффектов. Например, изменять значения переменных, внешних по отношению к этому фрагменту кода, или занимать память из «кучи».

Если побочный эффект имеется, то перед следующей итерацией он должен быть каким-то образом скомпенсирован. Это, конечно, влияет на точность таймирования, но обычно незначительно. А если два сравниваемых алгоритма имеют одинаковые побочные эффекты, этой погрешностью вообще можно пренебречь.

В приведённом ниже примере сравниваются два алгоритма, выполняющие одну и ту же задачу: генерация квадратов чисел от 0 до 999 и поиск среди этих значений некоторого числа. В первом случае для хранения значений используется контейнер vector, во втором — контейнер map.

Обратите внимание, что:

1) Таймируемый код слишком прост для моего процессора, поэтому для него используется цикл;
2) Таймируемый код имеет побочный эффект, который компенсируется вызовом метода clear() для контейнера.

Для результата вызова второй функции организована переменная, которая после таймирования выводится на экран.

#include <iostream>

```
#include <ctime>
#include <vector>
#include <map>
typedef std::vector<long> vecl;
typedef std::map<int, long> mapil;
const int MAX_ELEM = 1000;       // количество квадратов чисел
const long PATTERN = 999 * 999;  // это значение ищется среди квадратов чисел
const int MAX_TIMES = 1000;      // количество циклов при таймировании
void generator_vector(vecl& vec) {
   for (int i = 0; i < MAX_ELEM; i++) {
      vec.push_back(i * i);
   }
}
bool find_vector(const vecl& vec, long num) {
   for (auto it = vec.begin(); it != vec.end(); it++) {
      if (*it == num)
         return true;
   }
   return false;
}
void generator_map(mapil& mp) {
   for (int i = 0; i < MAX_ELEM; i++) {
      mp[i] = i * i;
   }
}
bool find_map(const mapil& mp, long num) {
   for (auto it = mp.begin(); it != mp.end(); it++) {
      if (it->second == num)
```

```
            return true;
        }
        return false;
    }
    int main() {
        vecl v;
        mapil m;
        clock_t the_time;
        bool res;
        double elapsed_time;
        std::cout << "Vector version" << std::endl;
        the_time = clock();
        for (int t = 0; t < MAX_TIMES; t++) {
            v.clear();
            generator_vector(v);
            res = find_vector(v, PATTERN);
        }
        elapsed_time = double(clock() - the_time) / CLOCKS_PER_SEC;
        std::cout << "Elapsed time " << elapsed_time << " sec." << std::endl;
        std::cout << "result =  " << res << "\n" << std::endl;
        std::cout << "Map version" << std::endl;
        the_time = clock();
        for (int t = 0; t < MAX_TIMES; t++) {
            m.clear();
            generator_map(m);
            res = find_map(m, PATTERN);
        }
        elapsed_time = double(clock() - the_time) / CLOCKS_PER_SEC;
        std::cout << "Elapsed time " << elapsed_time << " sec." << std::endl;
```

```
    std::cout << "result =  " << res << "\n" << std::endl;
    return 0;
}
```

Вышеописанный метод тестирования скорости выполнения фрагмента кода пригоден только в простейших случаях. Для более сложных случаев необходимо использовать профилировщик.

Использованные источники:

1. Аббакумов А.А., Акимов В.Л., Егунова А.И., Лещанкин К.А., Таланов В.М. Базы данных (MS ACCESS, MYSQL). Саранск: Изд-во Мордов. ун-та, 2011. С. 5.
2. Александров Э.Э., Афонин В.В. Программирование на языке С в Microsoft Visual Studio 2010 [Электронный ресурс]. Режим доступа: http://www.intuit.ru/department/pl/prcmsvs2010
3. Таненбаум Э. Современные операционные системы. СПб., 2010. 972 с.
4. Прата С. Язык программирования С. Лекции и упражнения. М.: Вильямс, 2006. 256 с.