**6 ЛАБОРАТОРНАЯ РАБОТА №6**

**«Исследование способов профилирования программного обеспечения»**

**6.1 Цель работы**

Исследовать критические по времени выполнения участки программного кода и возможности их устранения. Приобрести практические навыки анализа программ с помощью профайлера dotTrace.

**6.2 Вариант задания**

Разработать программу на основе библиотеки классов, реализованной и протестированной в предыдущей работе. Программа должна как можно более полно использовать функциональность класса. При необходимости для наглядности профилирования в методы класса следует искусственно внести задержку выполнения. Выполнить профилирование разработанной программы, выявить функции, на выполнение которых тратится наибольшее время. Модифицировать программу с целью оптимизации времени выполнения. Выполнить повторное профилирование программы, сравнить новые результаты и полученные ранее, сделать выводы.

**6.3 Ход выполнения работы**

6.3.1 В начале выполнения работы для класса, изменяющего амперсанты в строке на знаки вопроса, была добавлена операция для задержки выполнения, чтобы затем сравнить результаты изменения в скорости выполнения программы. Код класса для профилирования представлен в листинге 6.1.

Листинг 6.1 – Код класса для профилирования

namespace Lab3;

public class StringChanger

{

public string ChangeSymbols(string str)

{

Thread.Sleep(50);

if (str == null) throw new ArgumentNullException(nameof(str));

return str.Replace('?', '&');

}

}

Далее было выполнено профилирование в режиме tracing, его результаты отражены на рисунке 6.1.

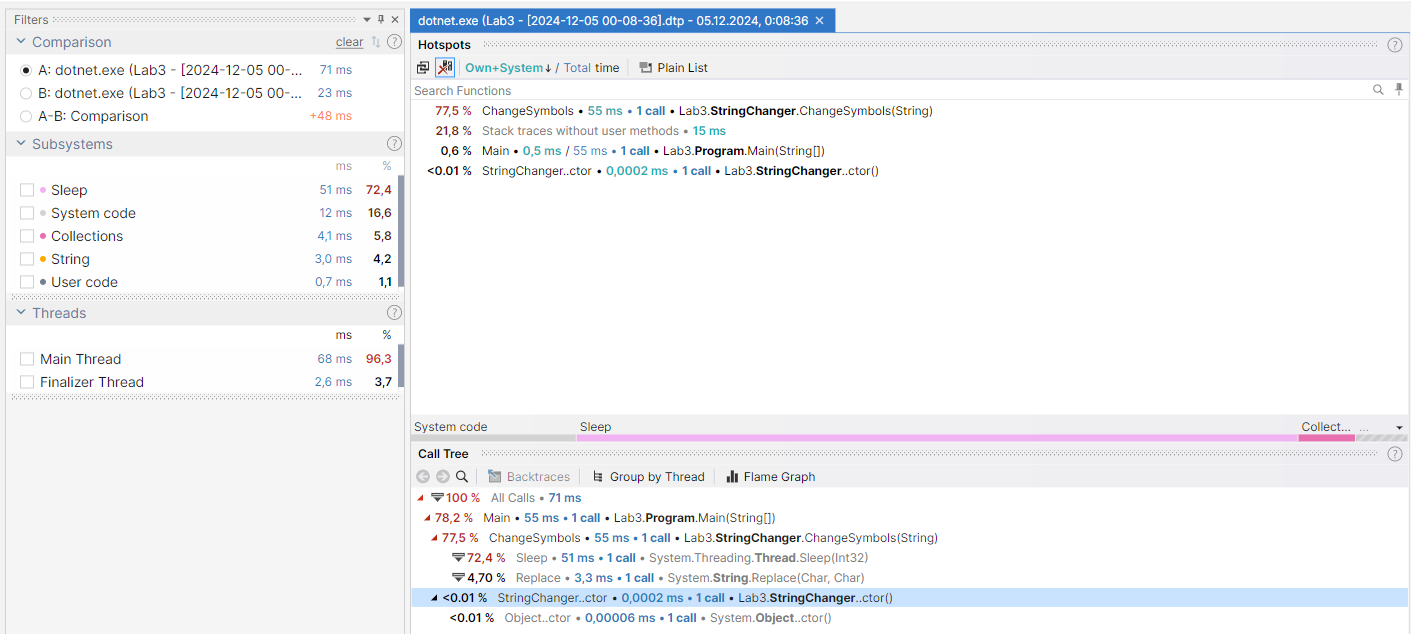


Рисунок 6.1 – Результаты профилирования первого варианта программы

В листинге 6.2 показан класс без внесенных изменений, который покажет реальное время выполнения программы без задержек.

Листинг 6.2 – Код класса для профилирования

namespace Lab3;

public class StringChanger

{

public string ChangeSymbols(string str)

{

if (str == null) throw new ArgumentNullException(nameof(str));

return str.Replace('?', '&');

}

}

На рисунке 6.2 показан результат профилирования для показанной выше программы.

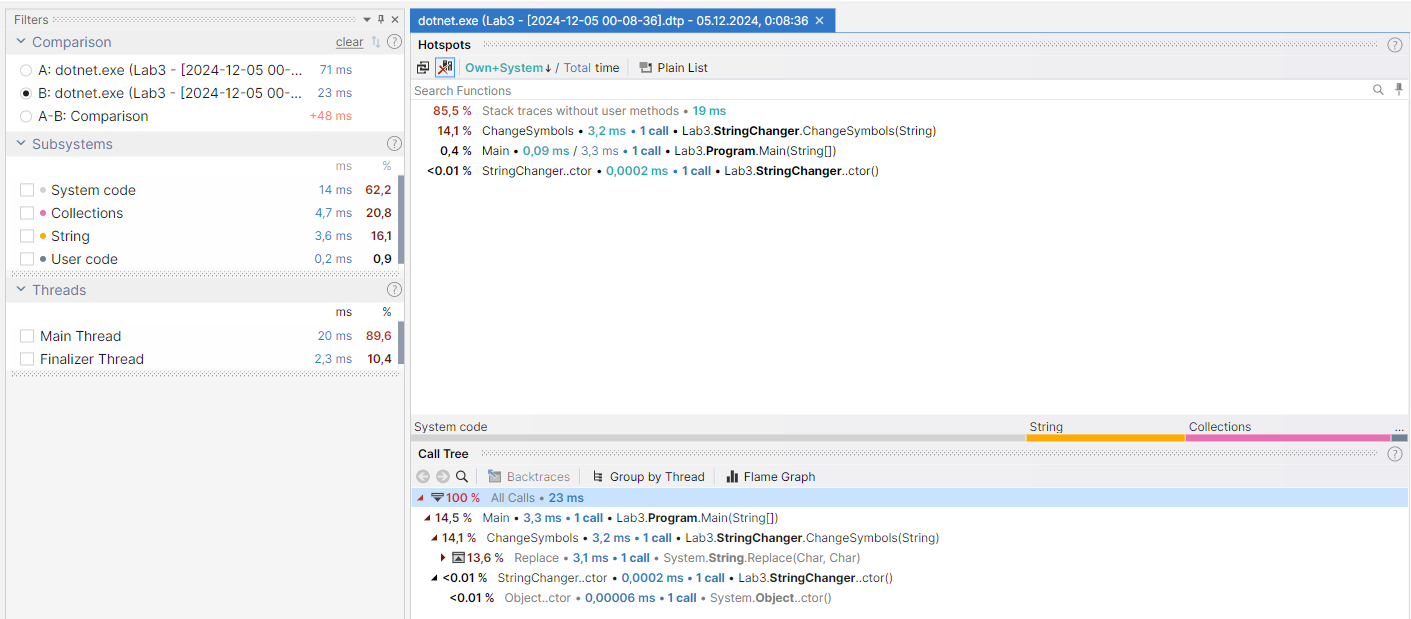


Рисунок 6.2 – Результаты профилирования второго варианта программы

Далее два полученных снимка были подвергнуты сравнению, полученные результат (рисунок 6.3) демонстрирует то, что первый вариант программы выполняется на 52 миллисекунды дольше, чем второй из-за использования метода выполнения паузы.

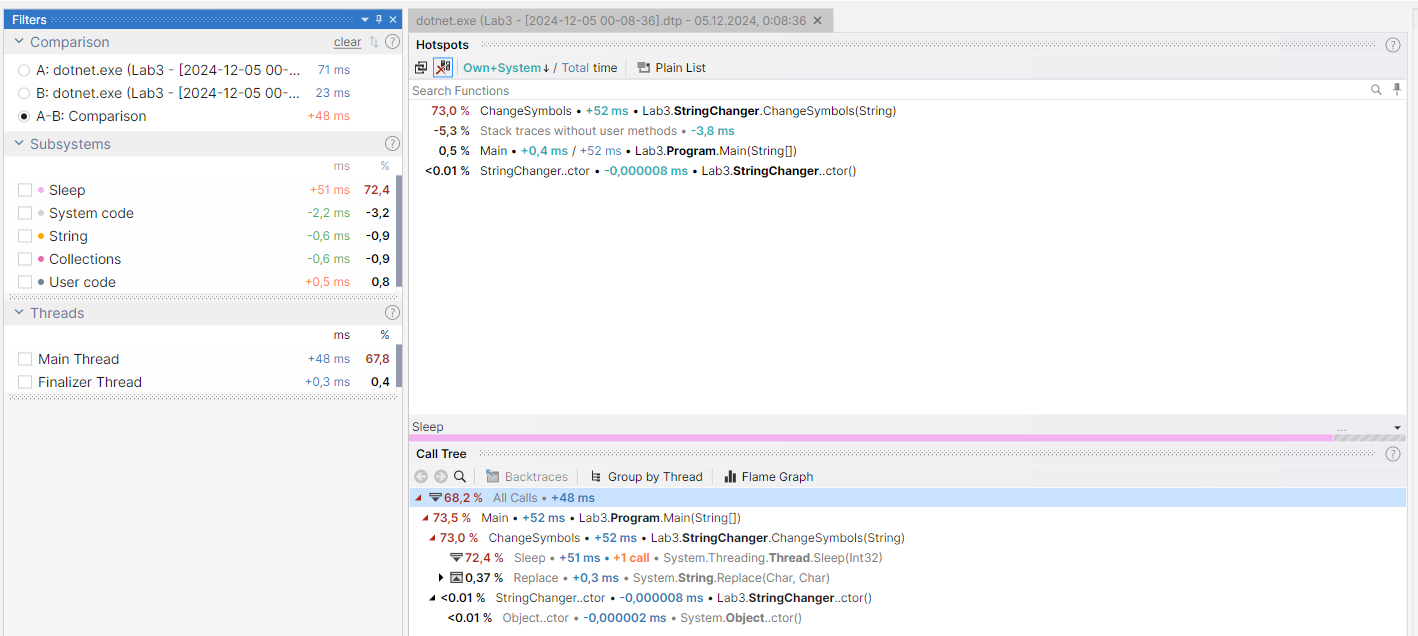


Рисунок 6.3 – Результат сравнения двух снимков

**Выводы**

В ходе лабораторной работы были исследованы критические по времени выполнения участки программного кода и возможности их устранения. Также были приобретены практические навыки анализа программ с помощью профайлера dotTracer. В конце выполнения лабораторной работы был написан отчет.