1. **Разберите программу, представленную в файле**[**task\_for\_lecture5.cpp**](https://canvas.instructure.com/courses/1844563/files/87817234/download?wrap=1)**. В программе создается 2 потока, каждый из которых вычисляет средние значения матрицы, один по строкам исходной матрицы***matrix***, а другой - по столбцам. Запустите программу и убедитесь в ее работоспособности.**

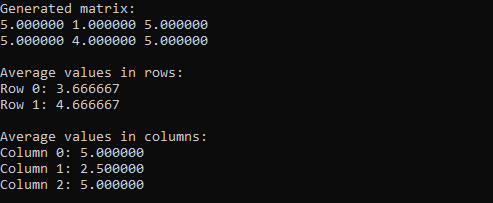


Рис.1 – Результат выполнения исходной программы

Вычисление средних значений каждой строки и каждого столбца:

Программа работает верно.

1. **Проанализируйте программу и введите в нее изменения, которые, по Вашему мнению, повысят ее производительность.**

Для оптимизации можно минимизировано количество используемых переменных. Например, мы можем создать для каждого цикла свой уникальный счётчик, но с точки зрения оптимизации – это трата ресурсов. Поэтому в частях программы, где не используется параллелизм можно использовать один счетчик для циклов for.

size\_t k = 0; //Наш счетчик для циклов for

size\_t m = 0; //Наш счетчик для внутренних циклов for

…

void InitMatrix(double\*\* matrix, const size\_t numb\_rows, const size\_t numb\_cols)

{

for (k = 0; k < numb\_rows; ++k)

{

for ( m = 0; m < numb\_cols; ++m)

{

matrix[k][m] = rand() % 5 + 1;

}

}

}

…

void PrintMatrix(double\*\* matrix, const size\_t numb\_rows, const size\_t numb\_cols)

{

printf("Generated matrix:\n");

for (k = 0; k < numb\_rows; ++k)

{

for (m = 0; m < numb\_cols; ++m)

{

printf("%lf ", matrix[k][m]);

}

printf("\n");

}

}

…

И так далее, во всех циклах for, которые не затрагивает параллелизм.

1. **Определите с помощью**Intel Parallel Inspector**наличие в программе таких ошибок как: *взаимная блокировка*, *гонка данных*, *утечка памяти*. Сделайте скрины результатов анализа**Parallel Inspector **(вкладки***Summary***,***Bottom-up***) для всех упомянутых ошибок*,*где отображаются обнаруженные ошибки, либо отражается их отсутствие. Запускайте анализы на разных уровнях (***Narrowest***,***Medium***,***Widest***).**

Проверка программы на проблемы с памятью.

На рисунке 2 показан результат анализа программы: в 4 местах обнаружена утечка памяти, т.е. память выделили, но не освободили.

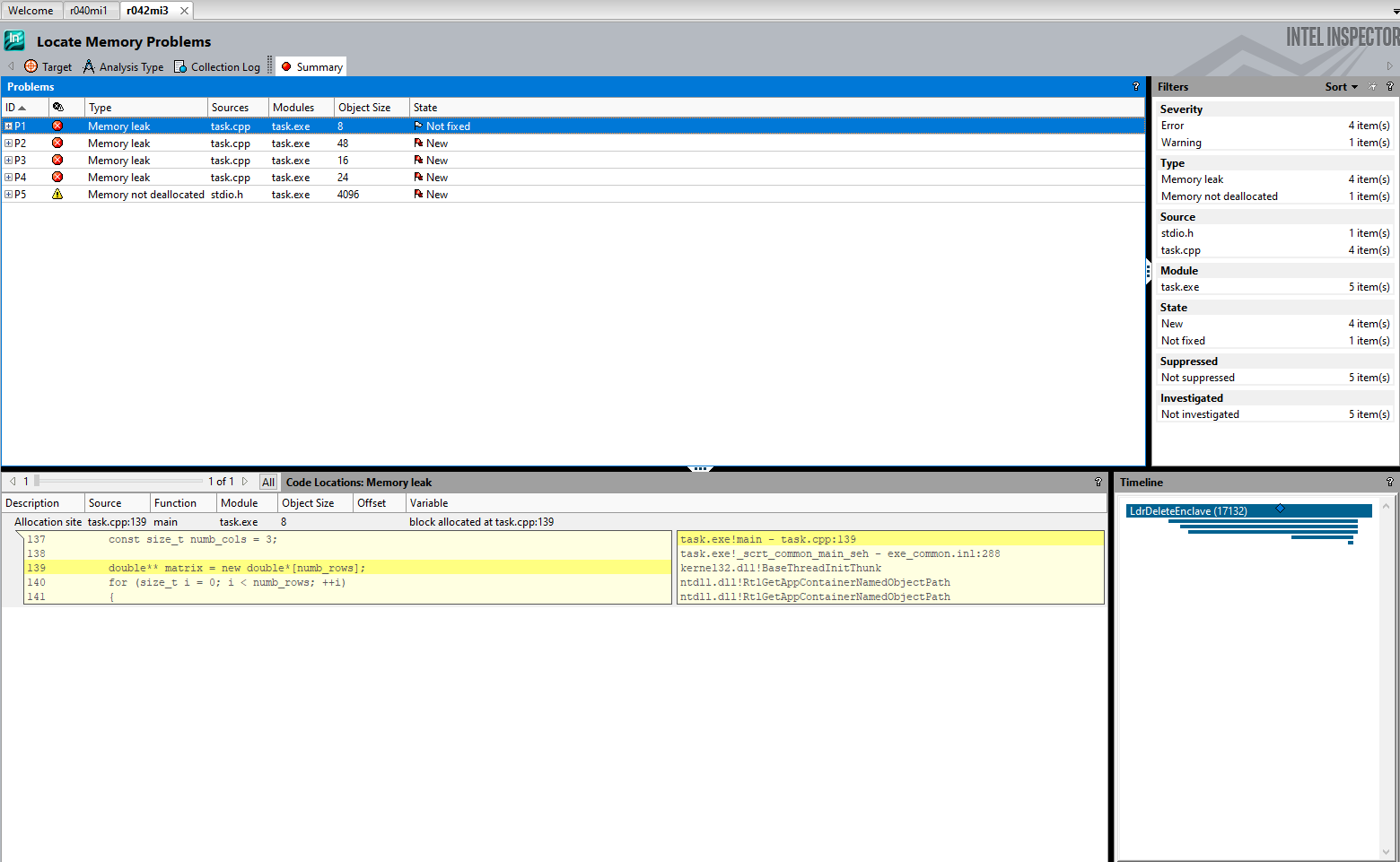


Рис.2 – Результат анализа программы на предмет ошибок, связанных с памятью

Проверка программы на гонки данных и взаимные блокировки.

На рисунке 3 показан результат анализа программы: гонки данных и взаимные блокировки не были обнаружены.

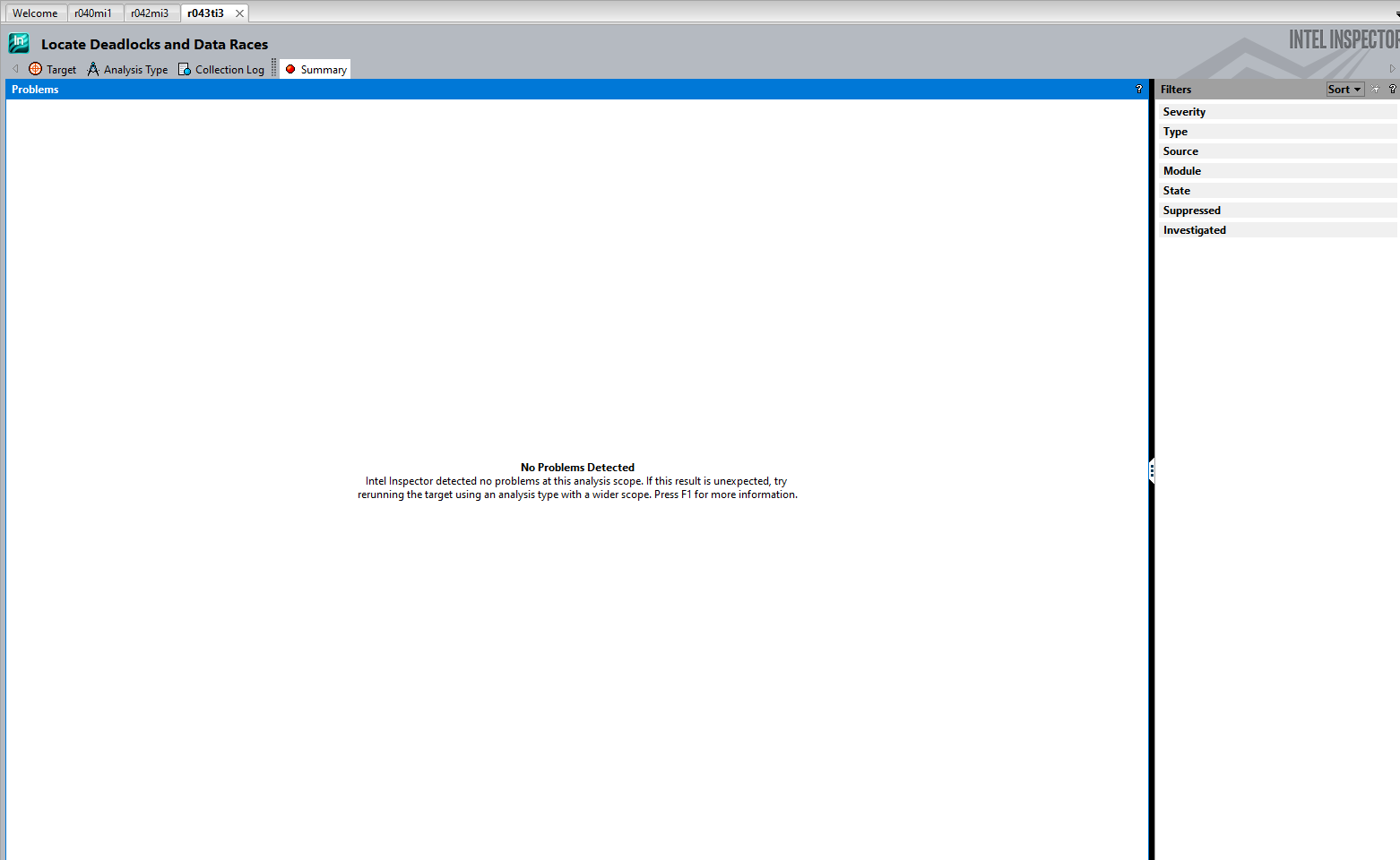


Рис.3 – Результат анализа программы на наличие гонок данных или взаимных блокировок

1. **Измените код программы таким образом, чтобы***Inspector***при проверке не находил в программе ошибок, перечисленных в**п. 3. **Сделайте скрины результатов запуска**Parallel Inspector**.**

Для устранения утечки памяти функция main была дополнена:

//Освобождение памяти--------------------

for (size\_t i = 0; i < numb\_rows; ++i)

{

delete[] matrix[i];

}

delete[]matrix;

delete[]average\_vals\_in\_cols;

delete[]average\_vals\_in\_rows;

//---------------------------------------

На рисунке 4 приведен результат анализа на предмет утечки памяти обновленной версии программы: утечки памяти не были выявлены.

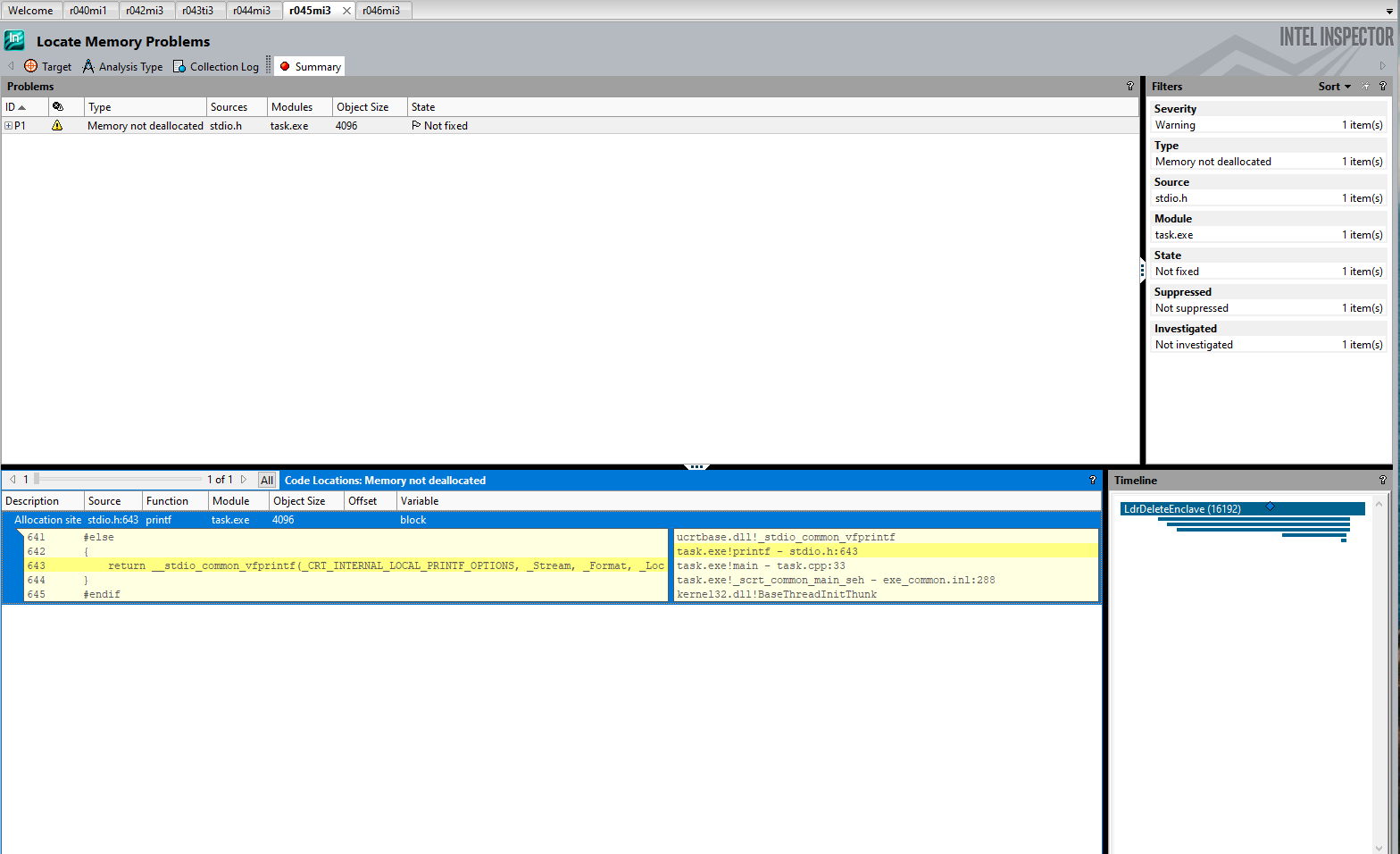


Рис.4 – Результат анализа программы на наличие утечек памяти

**Выводы:** В рамках данной лабораторной работы мы продолжали работу с инструментом Inspector. С помощью него мы провели анализ и обнаружили утечку памяти. Устранили утечку памяти путем освобождения выделенной памяти в конце программы.

При выполнении работы использовалась Visual Studio 2017 и IPS 2019.

Операционная система: Windows 10

Процессор: 2.80ГГц, 4 физических ядра, 8 логических ядер