# Цель и задачи лабораторной работы

**Цель:** научиться создавать многопоточные приложения с возможностью передачи параметров в параллельно выполняемые потоки.

**Задачи:**

1. Изучить механизм передачи параметров в поток;
2. Научиться решать практические задачи с использованием потоков с параметрами;
3. Изучить возможные методы передачи параметров.

# Реализация индивидуального задания

Согласно варианту задания, требуется с помощью параметризированных потоков выполнить метод возвращает матрицу случайных битовых значений (0 или 1), формируемую случайным образом, с форматом входных данных: количество строк и столбцов матрицы.

## Листинг программного кода

***namespace*** lab6

{

**using** System;

**using** System.Threading;

***class*** Program

    {

*// Класс для передачи параметров в поток*

**public** ***class*** ThreadParams

        {

**public** **int** StartRow { ***get***; ***set***; }

**public** **int** EndRow { ***get***; ***set***; }

**public** **int** Cols { ***get***; ***set***; }

**public** **int**[,] Matrix { ***get***; ***set***; }

**public** **int** ThreadId { ***get***; ***set***; }

**public** **int** Seed { ***get***; ***set***; } *// Для разных seed в каждом потоке*

        }

*// Метод, который будет выполняться в потоке*

**static** **void** GenerateMatrixPart(**object** data)

        {

            ThreadParams parameters **=** (ThreadParams)data;

            Random rand **=** **new**(parameters.Seed);

            Console.WriteLine($"Поток {parameters.ThreadId} начал генерацию строк {parameters.StartRow}-{parameters.EndRow}");

*for* (**int** i **=** parameters.StartRow; i **<=** parameters.EndRow; i**++**)

            {

*for* (**int** j **=** 0; j **<** parameters.Cols; j**++**)

                {

                    parameters.Matrix[i, j] **=** rand.Next(0, 2); *// 0 или 1*

                }

            }

            Console.WriteLine($"Поток {parameters.ThreadId} завершил генерацию");

        }

*// Основной метод для создания матрицы с использованием потоков*

**public** **static** **int**[,] CreateRandomBitMatrix(**int** rows, **int** cols, **int** threadCount)

        {

**int**[,] matrix **=** **new** **int**[rows, cols];

            Thread[] threads **=** **new** Thread[threadCount];

**int** rowsPerThread **=** (**int**)Math.Ceiling((**double**)rows **/** threadCount);

            Random seedGenerator **=** **new**();

*for* (**int** i **=** 0; i **<** threadCount; i**++**)

            {

**int** startRow **=** i **\*** rowsPerThread;

**int** endRow **=** Math.Min(startRow **+** rowsPerThread **-** 1, rows **-** 1);

*// Пропускаем, если строк для обработки нет*

*if* (startRow **>=** rows) *break*;

*// Создаем параметры для потока*

                ThreadParams parameters **=** **new**()

                {

                    StartRow **=** startRow,

                    EndRow **=** endRow,

                    Cols **=** cols,

                    Matrix **=** matrix,

                    ThreadId **=** i **+** 1,

                    Seed **=** seedGenerator.Next() *// Уникальный seed для каждого потока*

                };

*// Создаем и запускаем поток*

                threads[i] **=** **new** Thread(GenerateMatrixPart);

                threads[i].Start(parameters);

            }

*// Ожидаем завершения всех потоков*

*for* (**int** i **=** 0; i **<** threadCount; i**++**)

            {

*if* (threads[i] **!=** null **&&** threads[i].IsAlive)

                {

                    threads[i].Join();

                }

            }

*return* matrix;

        }

*// Вспомогательный метод для вывода матрицы*

**static** **void** PrintMatrix(**int**[,] matrix)

        {

**int** rows **=** matrix.GetLength(0);

**int** cols **=** matrix.GetLength(1);

            Console.WriteLine("\nСгенерированная матрица:");

*for* (**int** i **=** 0; i **<** rows; i**++**)

            {

*for* (**int** j **=** 0; j **<** cols; j**++**)

                {

                    Console.Write($"{matrix[i, j]} ");

                }

                Console.WriteLine();

            }

        }

*// // Статистика по матрице*

*// static void PrintMatrixStats(int[,] matrix)*

*// {*

*//     int rows = matrix.GetLength(0);*

*//     int cols = matrix.GetLength(1);*

*//     int ones = 0, zeros = 0;*

*//     for (int i = 0; i < rows; i++)*

*//     {*

*//         for (int j = 0; j < cols; j++)*

*//         {*

*//             if (matrix[i, j] == 1) ones++;*

*//             else zeros++;*

*//         }*

*//     }*

*//     int total = rows \* cols;*

*//     Console.WriteLine($"\nСтатистика матрицы:");*

*//     Console.WriteLine($"Единицы: {ones} ({ones \* 100.0 / total:F1}%)");*

*//     Console.WriteLine($"Нули: {zeros} ({zeros \* 100.0 / total:F1}%)");*

*//     Console.WriteLine($"Всего элементов: {total}");*

*// }*

**public** **static** **void** Main(**string**[] args)

        {

            Console.WriteLine("=== Генерация матрицы случайных битов с использованием потоков ===");

*// Параметры матрицы*

**int** rows **=** 6;

**int** cols **=** 8;

**int** threadCount **=** 3;

            Console.WriteLine($"Параметры: {rows}x{cols} матрица, {threadCount} потока");

*// Создаем матрицу*

**int**[,] matrix **=** CreateRandomBitMatrix(rows, cols, threadCount);

*// Выводим результаты*

            PrintMatrix(matrix);

*// PrintMatrixStats(matrix);*

        }

    }

}

## Описание кода

**Ключевые механизмы**

1. Класс для передачи параметров

* **Инкапсуляция параметров** в отдельный класс
* **Уникальный Seed** для каждого потока (решает проблему одинаковых случайных последовательностей)

2. Параметризованные потоки

* Использование **ParameterizedThreadStart** делегата
* Передача данных через объект parameters

3. Запуск потоков с параметрами

* Явная передача параметров в метод Start()
* Каждый поток получает свой диапазон строк

**4. Генерация бинарной матрицы**

* Заполнение **0 и 1** (rand.Next(0, 2))
* Матрица размером **6×8** обрабатывается **3 потоками**
* **Равномерное распределение** работы между потоками

## Результат работы программы

=== Генерация матрицы случайных битов с использованием потоков ===

Параметры: 6x8 матрица, 3 потока

Поток 3 начал генерацию строк 4-5

Поток 1 начал генерацию строк 0-1

Поток 2 начал генерацию строк 2-3

Поток 1 завершил генерацию

Поток 3 завершил генерацию

Поток 2 завершил генерацию

Сгенерированная матрица:

0 1 1 0 1 1 1 1

1 0 0 1 1 1 1 1

0 1 1 1 0 0 0 0

0 1 1 0 1 0 1 1

1 1 0 0 1 0 0 0

1 0 1 0 0 1 1 1

# Контрольные вопросы

1. **Опишите возможные пути передачи параметров в поток.**
   1. ParameterizedThreadStart
   2. Лямбда-выражения
   3. Класс-контейнер
   4. Замыкания (captured variables)
2. **Подумайте, можно ли передать в несколько потоков один и тот же параметр (ссылку на объект)? Ответ обоснуйте.**
   1. Да, можно, но с осторожностью
      1. Могут возникнуть проблемы, т. к. если объект изменяется из нескольких потоков, то могут возникнуть состояния гонки (race conditions).
      2. Если объект предназначен только для чтения, то проблем не будет.