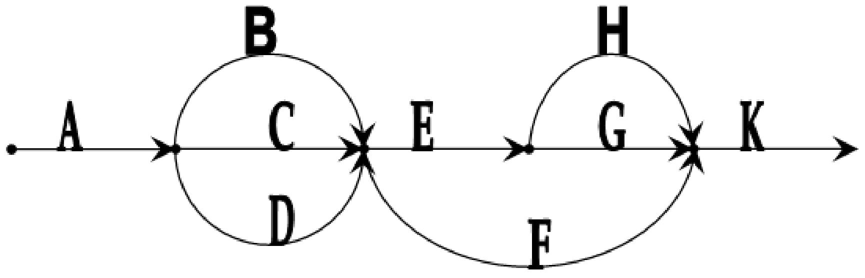
1. Task Description

Need to create a program that implements the parallel execution of several tasks, each of which solves some given function. A graph of tasks is given, i.e. the sequence of their execution is determined, as well as the need for parallel execution of some tasks relative to each other:



According to the type of graph, some tasks can initiate the execution of others. Tasks can use common (shared) objects and data, therefore, in the designed program, it is necessary to provide synchronization of access to such objects and data.

Files Created:

Program.cs - the main entry point for the application

TaskThread.cs – abstract thread class

SeeThreads.cs – functions for button clicks, thread classes A - K, thread initialization and start

Locker.cs – lock object class

Logger.cs – class for writing information to a log file

2. Description of the program and implementation means

Programming language: C#.

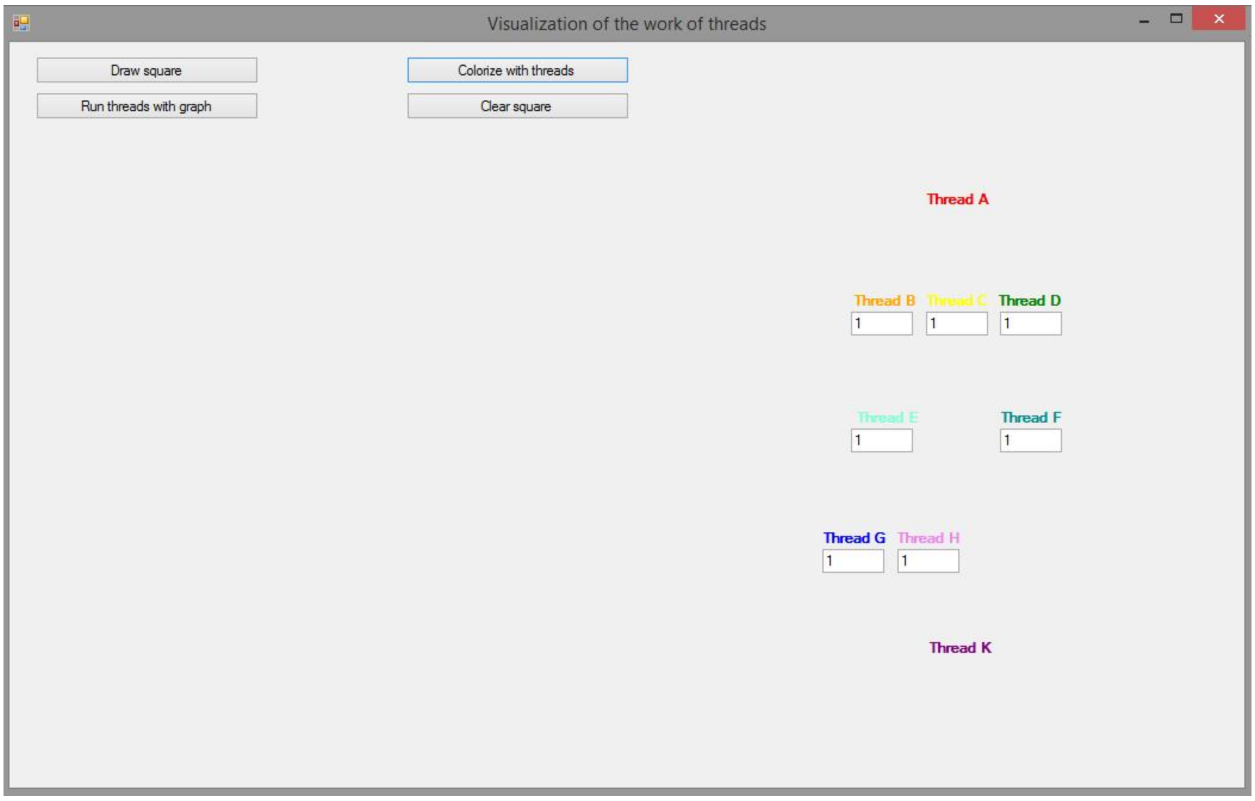
Application type: Windows Form.

Each task (A, B, ...) is framed as a thread class ‘Thread’. Each task performs a certain operation on the array elements (parallel threads perform different actions on the same array elements). Array access synchronization is implemented using the lock operator ‘lock’.

The lock operator acquires a mutually exclusive lock on the given object before executing certain statements, and then releases the lock. During a lock, the thread holding the lock can acquire and release the lock again. Any other thread cannot acquire the lock and is waiting for it to be released.

Program description

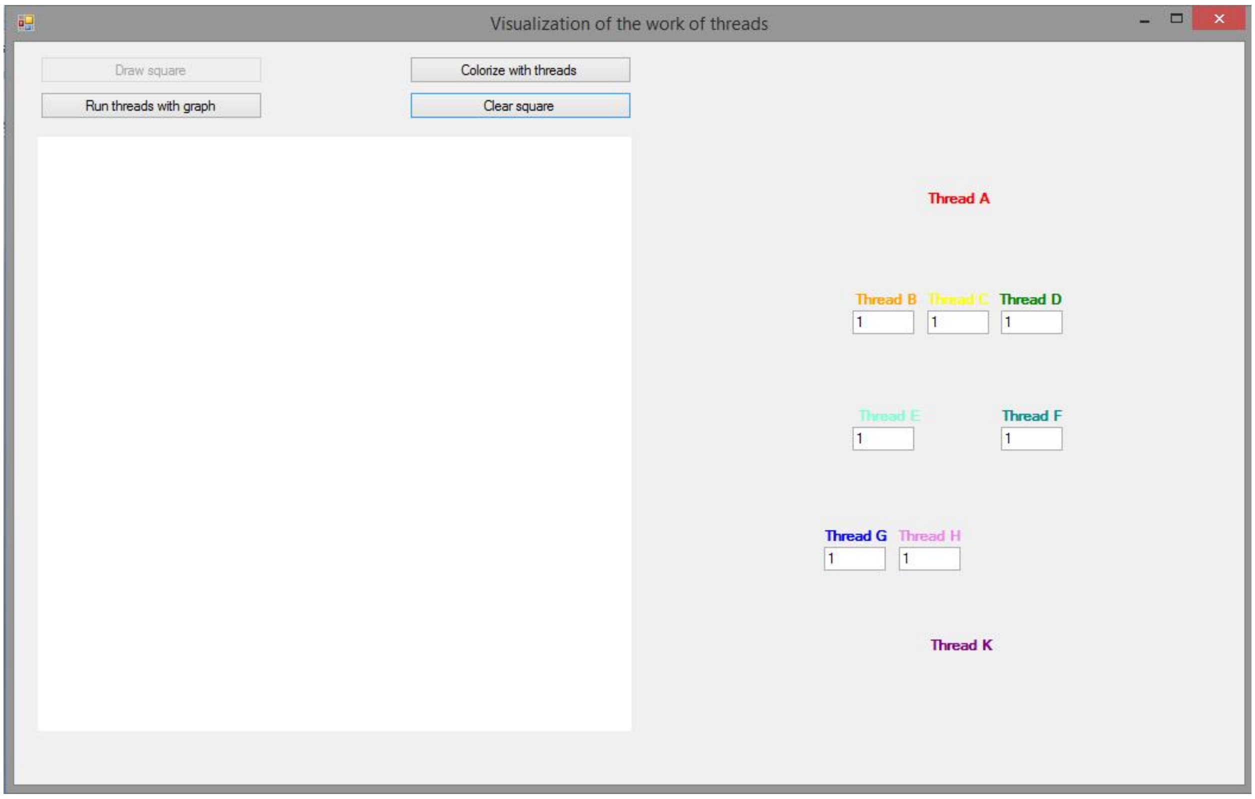
The program implements a graphical array with a size of 50 * 50 cells. Initial view of the form:



There are 4 buttons and a threads graph on the form.

When the form is initialized, two arrays are initialized: the array of colors 'arr' (white, red, yellow, etc.) and the array of graphic elements (squares) PictureBox. Both arrays are 50 * 50 elements.

When the "Draw square" button is pressed, all elements of the array arr are assigned the color value = "white", in the loop all elements of the graphic array are assigned the value of the property BackColor = White (coloring the square in white), and the position of each element of the array (square) is determined on the form. Result:



When you click the 'Run threads with graph' button, the work of parallel threads is implemented. Each thread in the loop assigns a color to the array element 'arr' in the given interval:

Thread A – assigns value «Red» to elements 0 – 500;

Thread B – assigns value «Orange» to elements 500 – 1000;

Thread C – assigns value «Yellow» to elements 500 – 1000;

Thread D – assigns value «Green» to elements 500 – 1000;

Thread E – assigns value «Aquamarine» to elements 1000 – 1500;

Thread F – assigns value «DarkCyan» to elements 1000 – 2000;

Thread G – assigns value «Blue» to elements 1500 – 2000;
Thread H – assigns value «Violet» to elements 1500 – 2000;
Thread K – assigns value «Purple» to elements 2000 – 2500;

All threads are initialized and connected in a thread graph. To start the graph, the first thread A is run. Further, the threads work according to the associated graph.

Threads B, C, D work in parallel and assign their color value to the same elements of the 'arr' array, threads E and F, and threads G, H and F also work in parallel (threads are started according to the graph) and color the same elements 'arr' array. The resulting array element color is determined by the thread that last accessed it.

Also, all the actions of the threads are recorded in the Log file (task execution protocol) - including in parallel mode. Each line in the Log file contains information about the recording of the stream of its color in the cell and the time of this recording. Access to the file is synchronized for parallel threads, as is access to the 'arr' array. When writing to the Log file, the existence of the file is checked. If the file is not found, then it is created.

To synchronize access to the elements of the array and the Log file, the lock operator 'lock' is used. For locking, the class 'Locker' was created, containing one property 'Counter'.

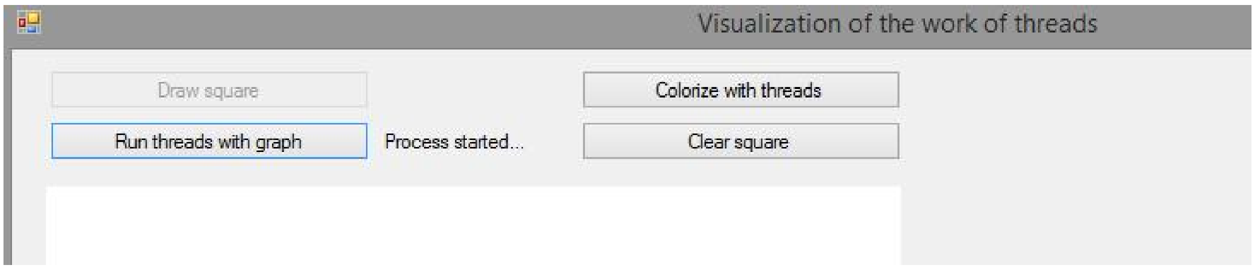
An abstract class has been created for threads, and every thread is a child of it. The abstract class has a Run() method that starts a thread. The method is implemented as follows:

- at the beginning of the method the counter is blocked and incremented,
- then some actions are performed by the thread,
- at the end, the counter is blocked again and decremented, after which a check is performed: if the counter = 0, i.e. all parallel threads have completed their work, as well as completed threads - not the last ones in the graph, then the next bunch of parallel threads is launched.

Each thread in the loop paints one of the elements of the array in a specific color. Since the number of elements is relatively small and each thread runs too fast, not having time to transfer access to another parallel thread, an artificial slowdown of threads was introduced. Each element of the array is colored by the stream number of times = the number of ‘Wait’ entered by the user * 1000000. For each thread ‘Wait’ value = 1. Thus, the conditional execution duration of each thread is entered as input.

To check the completion of all threads, a timer was added to the program, which starts when the "Start threads with graph" button is pressed and stops after setting the stop variable to -1 at the end of the thread K. The timer is necessary for checking, because all work on the array elements occurs in other, non-control threads.

When the threads start running, the text ‘Process started’ is displayed:



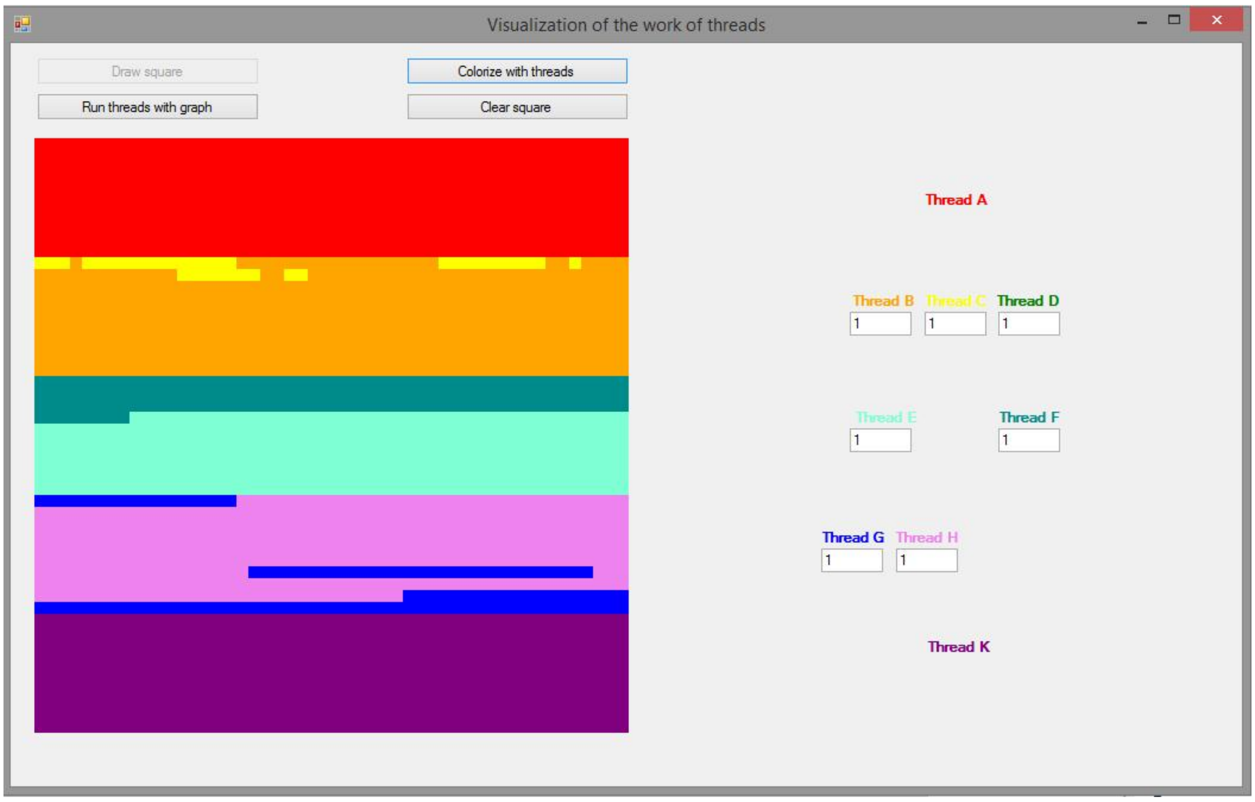
Process started...

After all threads A - K have completed their work, this inscription disappears.

After the threads finished, all elements of the ‘arr’ array were filled with colors: elements 0 - 500 are filled with red in thread A, elements 500 - 1000 are filled with either orange in thread B, or yellow - in thread C, or green - in thread D, depending on which thread accessed the element last. Elements 1000 - 1500 are filled with either cyan in the E stream or turquoise in the F stream, elements 1500 - 2000 are either also turquoise in the F stream, or blue in the G stream, or lilac in the H stream. Elements 2000 - 2500 are filled only with purple in the flow K.

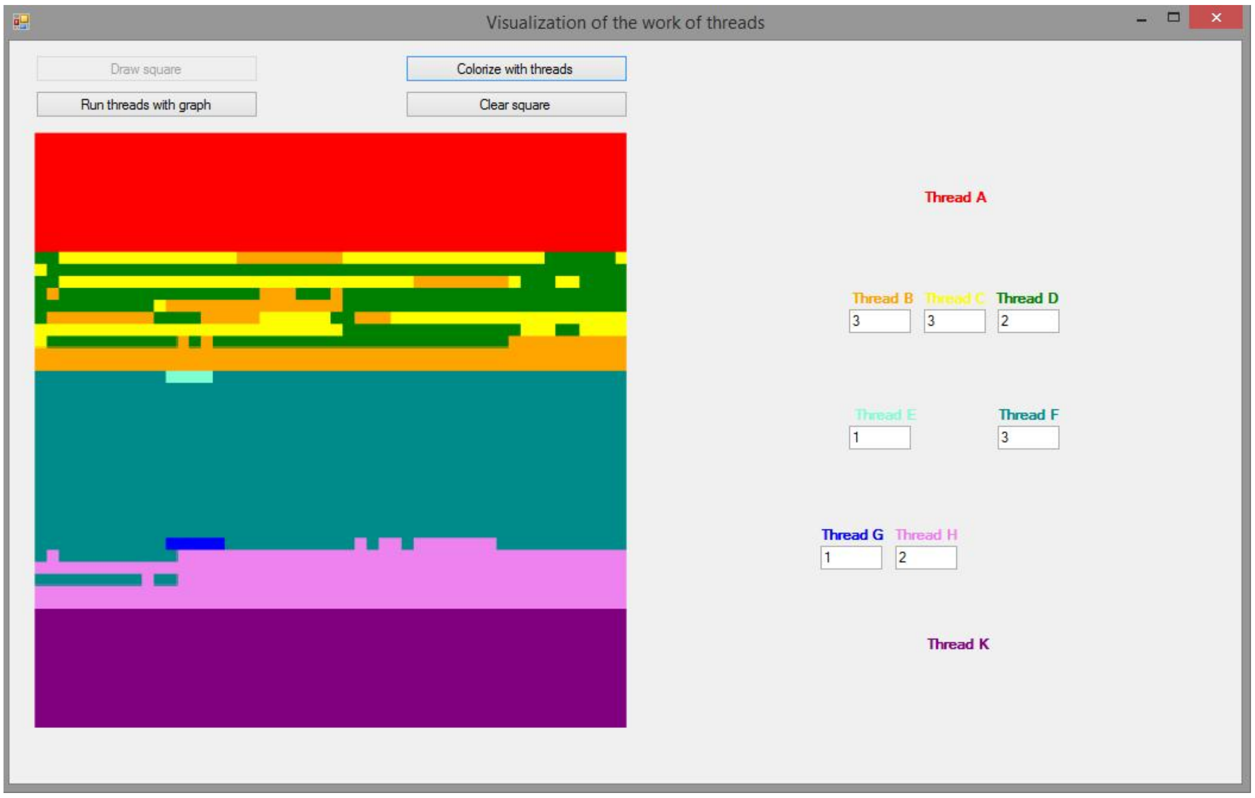
When you click the ‘Colorize with threads’ button, the elements of the PictureBox graphic array (more precisely, their BackColor property) are assigned the color value of the element of the same array number arr that was filled in threads A - K. As a result, you can visually see the work of the threads.

Examples of program execution - on the resulting square (graphic array), it can be seen that the threads pass access to the array elements in turn:



1 1 1
1 1
1 1

Thread duration can be changed:



3 3 2
1 3
1 2

When the ‘Clear square’ button is pressed, all elements of the arr array are filled with white, all elements of the graphical array are also painted white.