ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | И.Д. Свеженин |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2 |
| Многопоточное программирование |
| по курсу: Кроссплатформенное программирование |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4329 |  |  |  | Д.С. Шаповалова |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

Содержание

[1. Цель работы: 3](#_Toc210205481)

[2. Задание: 3](#_Toc210205482)

[4. Скриншоты, иллюстрирующие результаты работы программы: 5](#_Toc210205483)

[5. Вывод: 6](#_Toc210205484)

# **1. Цель работы:**

Изучение и практическое применение многопоточного программирования с использованием синтаксиса и возможностей языка высокого уровня Kotlin.

# **2. Задание:**

**Вариант 17**

Напишите программу, которая каждую секунду отображает на экране данные о времени, прошедшем от начала сессии, а другой её поток выводит сообщение каждые 5 секунд. Предусмотрите возможность ежесекундного оповещения потока, воспроизводящего сообщение, потоком, отсчитывающим время. Не внося изменений в код потока-"хронометра", добавьте ещё один поток, который выводит на экран другое сообщение каждые 7 секунд.

**3. Краткое описание хода разработки, алгоритма работы программы и назначение используемых технологий**

Ход разработки:

1. Разработаем поток хронометра.
2. Разработаем поток «вывода сообщения раз в 5 секунд».
3. Повторим пункт 2 для 7 секунд.

**Используемые технологии:**

* Kotlin – используется для работы написанного кода.
* Thread – реализация потоков.
* Synchronized (monitor) – Механизм синхронизации – пока 1 поток её держит, другие не выполняют действия из их аналогичного блока.

**Описание алгоритма работы программы:**

1. Запускаем 3 потока выполняться одновременно.
2. Один из потоков вывода сообщения захватывает monitor (механизм синхронизации) и сразу отпускает, ожидая команды от какого-нибудь другого потока.
3. Второй из потоков вывода сообщения повторяет.
4. В это же время, хронометр отсчитывает 1 секунду и записывает, что прошла 1 секунда и пишет о том, что стало на 1 секунду больше секунд, чем было 1 секунду назад.
5. Освободившийся монитор захватывает хронометр, и даёт сигнал другим потокам действовать.
6. Какой-либо поток вывода сообщения просыпается, записывает себе прошедшую 1 секунду, проверяет, прошло ли уже n секунд с того момента, как он запустился впервые или вывел сообщение о n-ной секунде.
7. Поток начинает цикл заново и снова засыпает.
8. Другой поток вывода сообщения действует аналогично.
9. Повторять с 4 пункта.

# **4. Скриншоты, иллюстрирующие результаты работы программы:**

В качестве демонстрации работы программы приведём несколько скриншотов с разными значениями х и точностью.

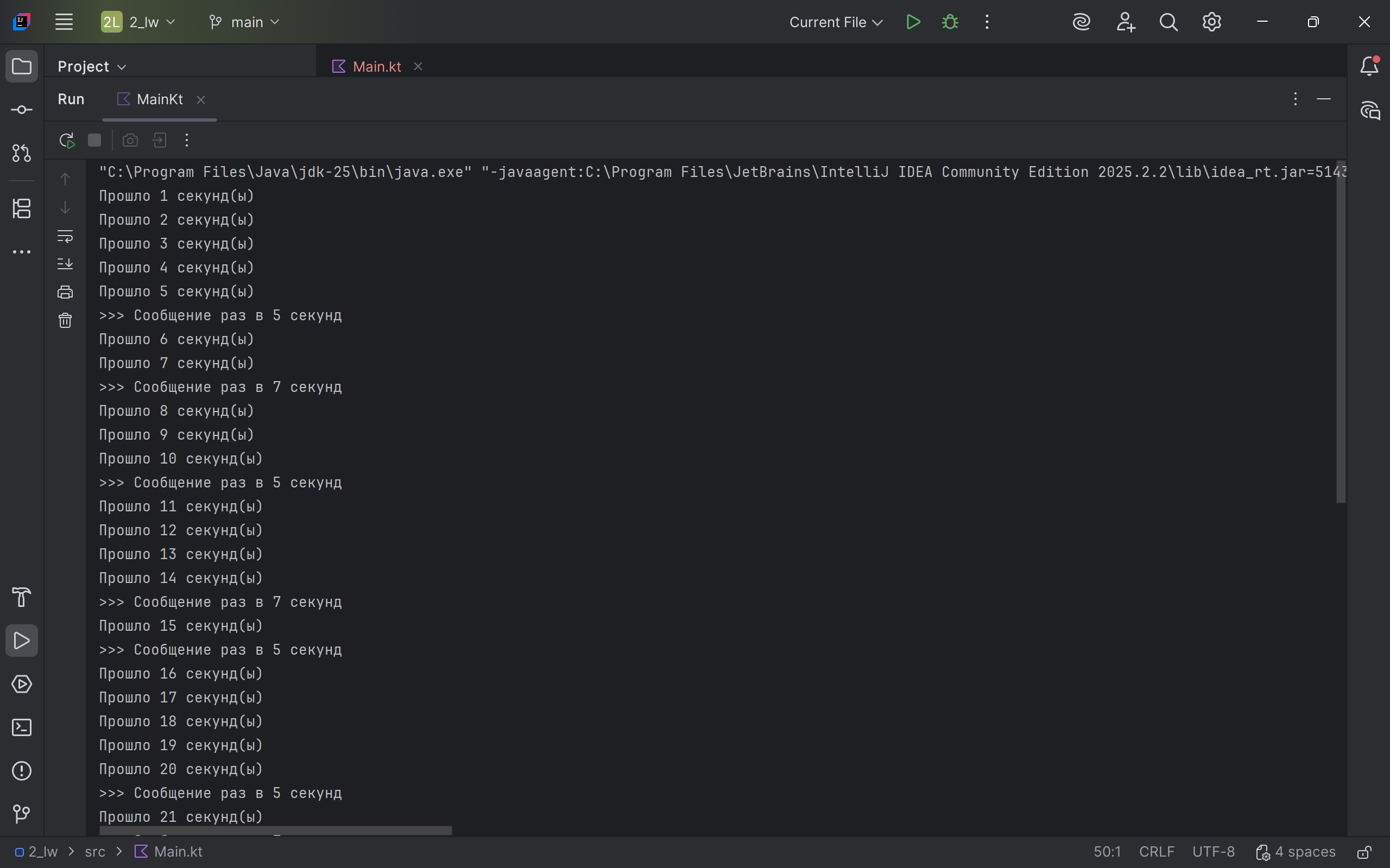


Рисунок 2.1 – Результат работы программы

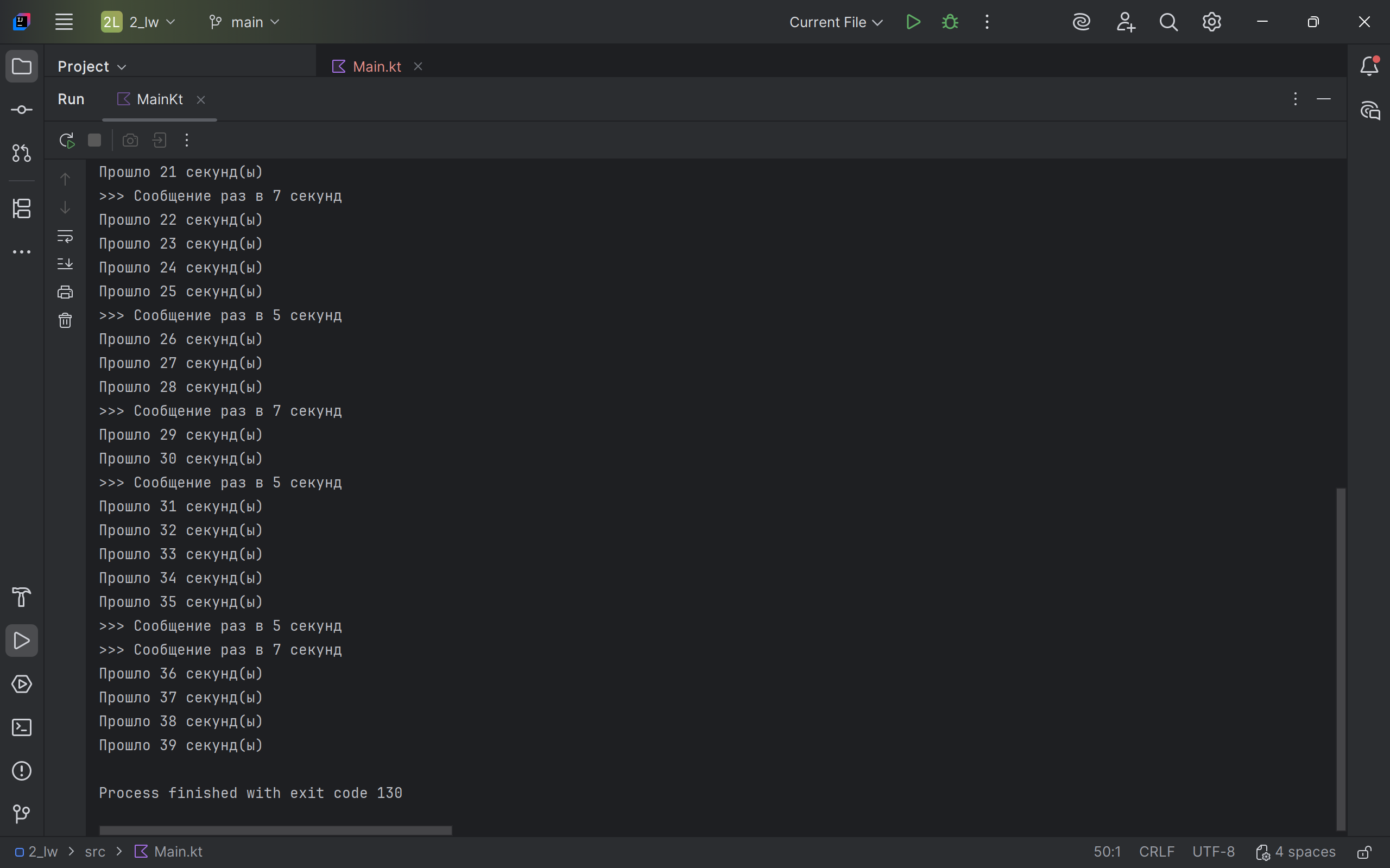


Рисунок 2.2 – Результат работы программы

# **5. Вывод:**

В данной работе была изучена концепция многопоточного программирования, используя язык высокого уровня – Kotlin. Была написана программа, каждую секунду выводящая сообщение о том, что прошло на 1 секунду больше, чем 1 секунду назад, сообщение о том, что прошло 5 секунд и 7 секунд.

ПРИЛОЖЕНИЕ А

Листинг программы:

fun main() {  
 val monitor = Object()  
 var seconds = 0  
  
 // Хронометр  
 val timerThread = Thread **{** while (true) {  
 Thread.sleep(1000)  
 seconds++  
 *println*("Прошло $seconds секунд(ы)")  
 *synchronized*(monitor) **{** monitor.notifyAll() // оповещаем остальные потоки  
 **}** }  
 **}** // Поток с сообщением раз в 5 секунд  
 val fiveSecThread = Thread **{** var localSeconds = 0  
 while (true) {  
 *synchronized*(monitor) **{** // захватываем монитор  
 monitor.wait() // отпускаем монитор, засыпаем  
 localSeconds++  
 if (localSeconds % 5 == 0) {  
 *println*(">>> Сообщение раз в 5 секунд")  
 }  
 **}** }  
 **}** // Поток с сообщением раз в 7 секунд  
 val sevenSecThread = Thread **{** var localSeconds = 0  
 while (true) {  
 *synchronized*(monitor) **{** monitor.wait()  
 localSeconds++  
 if (localSeconds % 7 == 0) {  
 *println*(">>> Сообщение раз в 7 секунд")  
 }  
 **}** }  
 **}** // Запуск  
 timerThread.start()  
 fiveSecThread.start()  
 sevenSecThread.start()  
}