Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 4

по курсу: «Моделирование»

Тема Моделирование работы системы массового обслуживания

Вариант 20 (4)

Студент Якуба Д. В.

Группа ИУ7-73Б

Оценка (баллы) _____

Преподаватель Рудаков И.В.

Москва, 2021

1. Задание

Смоделировать работу системы, состоящей из генератора, очереди и обслуживающего аппарата. Генерация заявок происходит по закону равномерного распределения с заданными параметрами a, b. Обработка заявок происходит по закону распределения Гаусса с заданными параметрами μ, σ.

Требуется определить длину очереди, при которой не будет потери сообщений.

Также смоделировать работу системы с построенной обратной связью, в качестве параметра используется процент обработанных заявок, вновь поступивших на обработку.

Протяжка модельного времени должна осуществляться по Δt и по событийному принципу. Обозначить, есть ли разница в результатах.

2. Теория

2.1 Система массового обслуживания

СМО – это система, которая производит обслуживание поступающих в неё требований. Обслуживание требований в СМО осуществляется обслуживающими аппаратами. Классическая СМО содержит в себе от одного до бесконечного числа подобных аппаратов.

2.2 Протяжка модельного времени по Δt

Принцип Δt заключается в последовательном анализе состояний всех элементов системы в некоторый момент времени $t + \Delta t$ по заданному состоянию этих элементов в момент времени t. При этом новое состояние элементов определяется в соответствии с их алгоритмическим описанием с учётом действующих случайных факторов, задаваемых распределениями вероятности. В результате такого анализа принимается решение о том, какие общесистемные события должны имитироваться программной моделью на текущий момент времени.

2.3 Протяжка модельного времени по событийному принципу

Характерным свойством систем обработки информации является тот факт, что состояние отдельных элементов изменяется в некоторые дискретные моменты времени, совпадающие с моментами времени поступления сообщений в систему и так далее. Моделирование и продвижение времени в системе посему так же удобно проводить, используя событийный принцип, при котором состояние всех элементов имитационной модели анализируется лишь в момент появления какого-либо события. Момент поступления следующего события определяется минимальным значением из списка будущих событий, представляющего собой совокупность моментов ближайшего изменения состояния каждого из элементов системы.

3. Выполнение

3.1 Замечания

В качестве фактора окончания проведения моделирования в данной работе принято окончание обработки всех поступивших и вновь поступивших заявок.

Данный выбор обусловлен тем, что в реальной задаче моделирования подобной системы могло бы потребоваться определить, например, время обработки обслуживающим аппаратом 1000 первично поступивших заявок, каждая из которых с некоторым шансом может вновь вернуться в очередь.

В дальнейшем, при предоставлении результатов, будет заметно, что из 100 поданных на вход заявок, которые должны возвращаться после обработки в очередь с вероятность 0.3, вернётся более 30% (что было бы ожидаемым результатом, ведь генерация происходит по закону равномерного распределения), так как подобные заявки также имеют шанс вновь вернуться в очередь.

3.2 Интерфейс разработанного приложения

На рисунке 3.1 предоставлен интерфейс разработанного приложения.

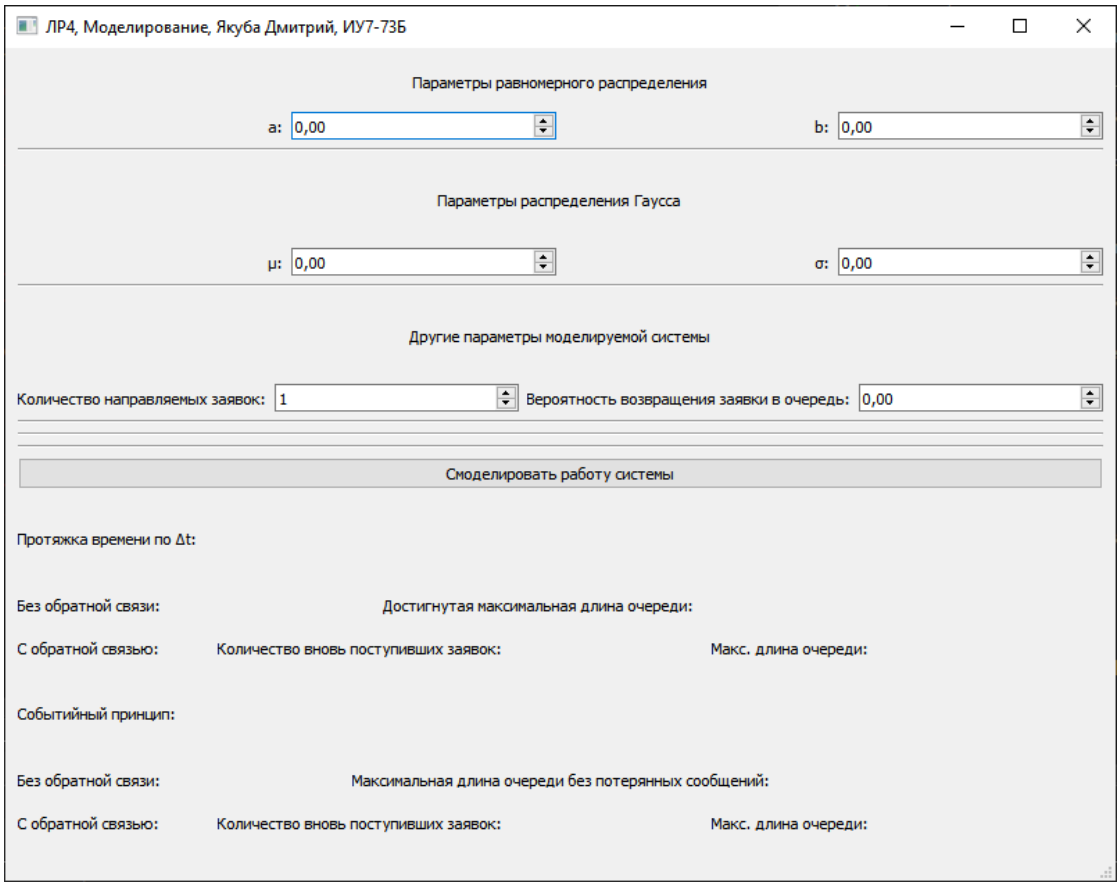

Рис. 3.1, Интерфейс приложения

3.3 Примеры работы

На рисунках 3.2-3.5 предоставлены примеры работы приложения.

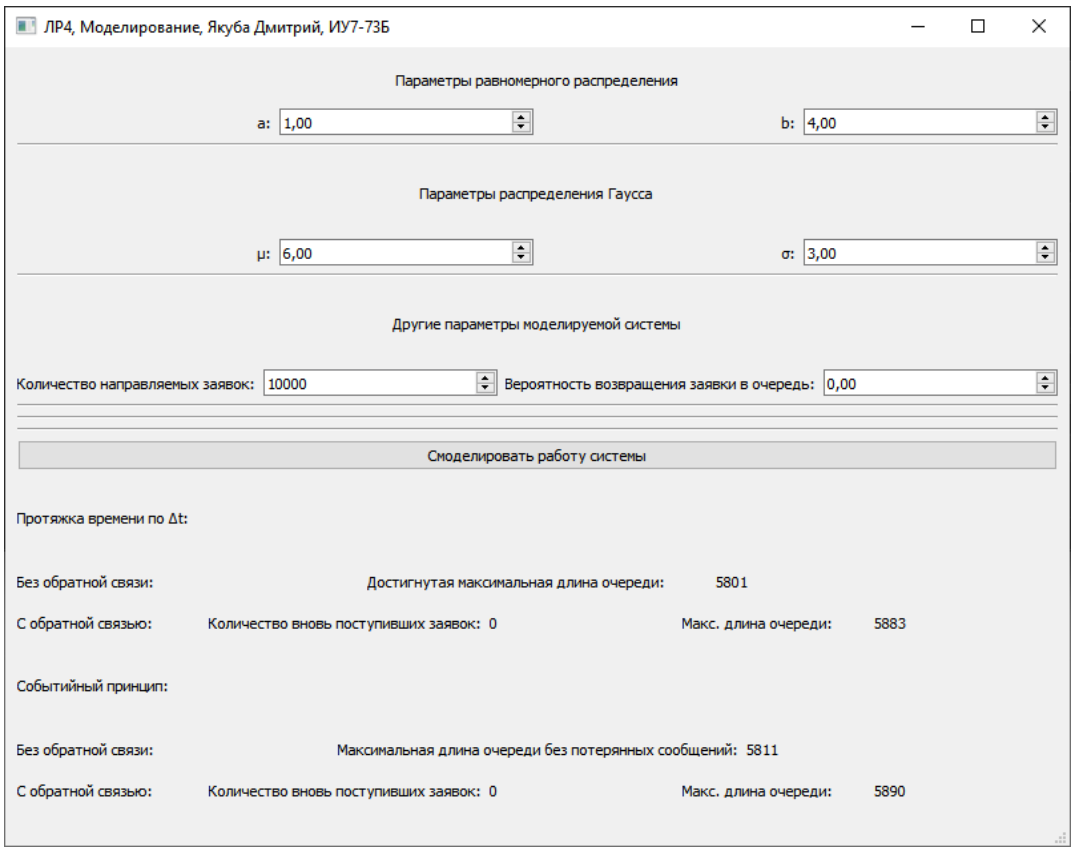

Рис. 3.2, Пример работы для системы без обратной связи с ОА, обрабатывающим запросы медленнее их генерации

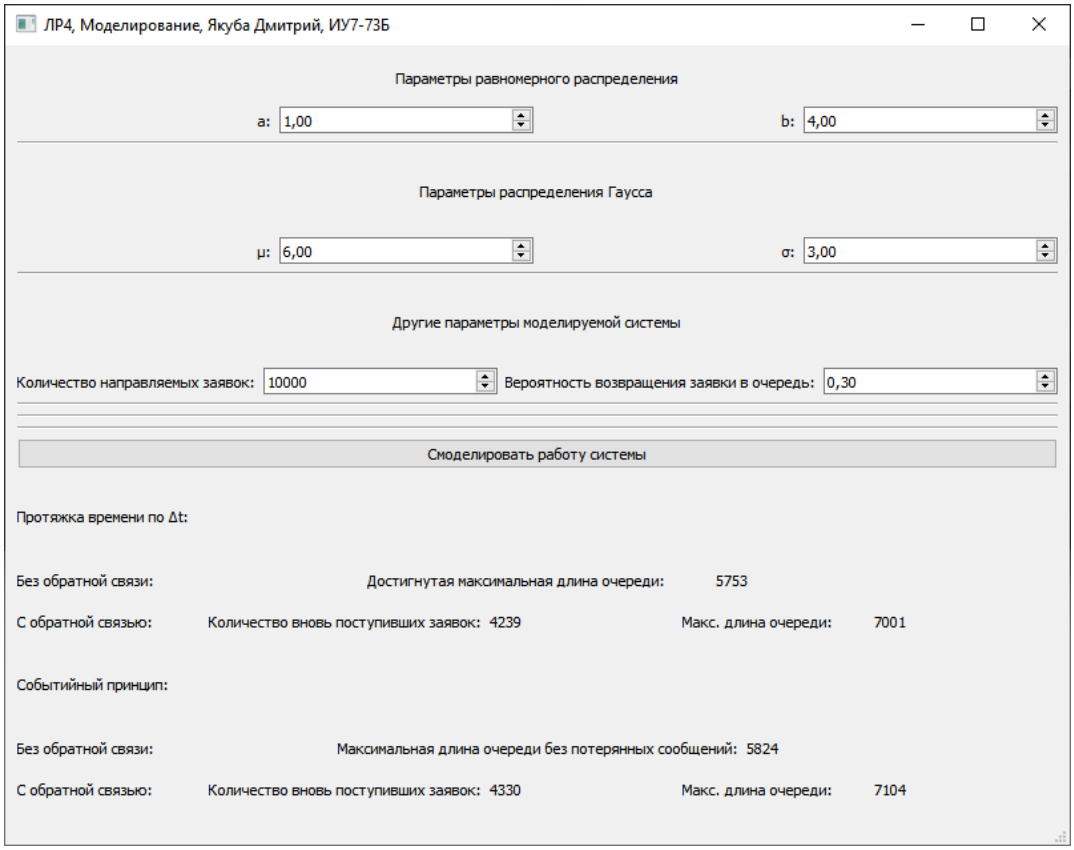

Рис. 3.3, пример работы для системы с обратной связью с ОА, обрабатывающим запросы медленнее их генерации

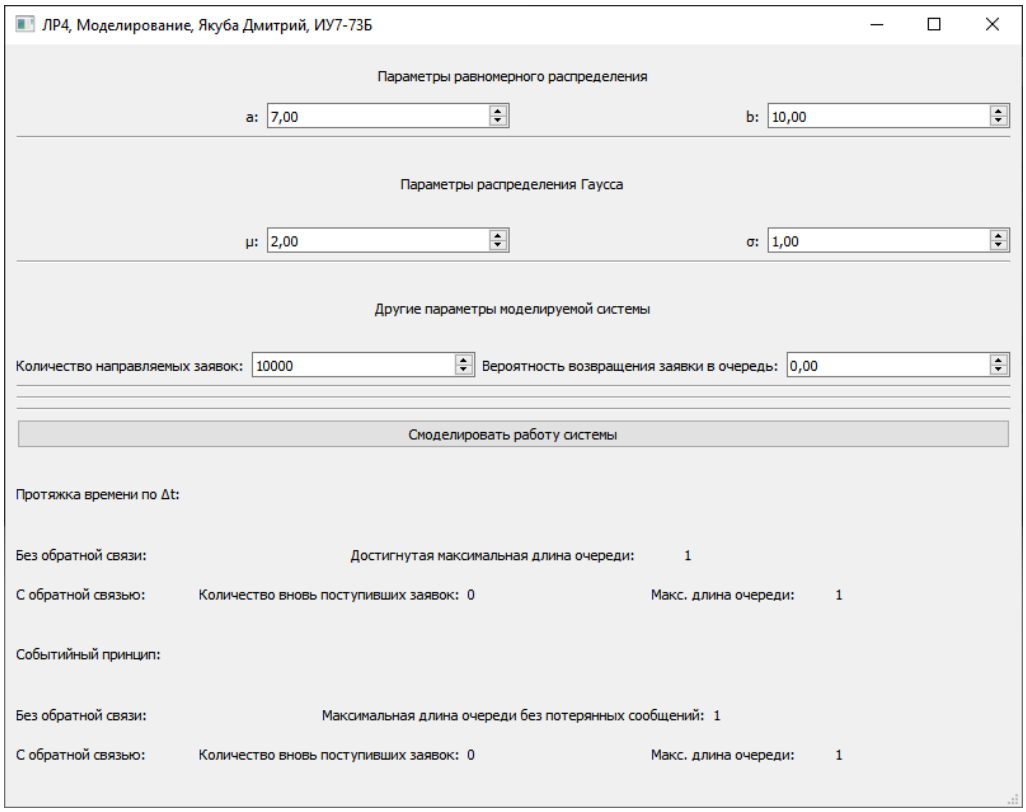

Рис. 3.4, Пример работы для системы без обратной связи с ОА, обрабатывающим запросы быстрее их генерации

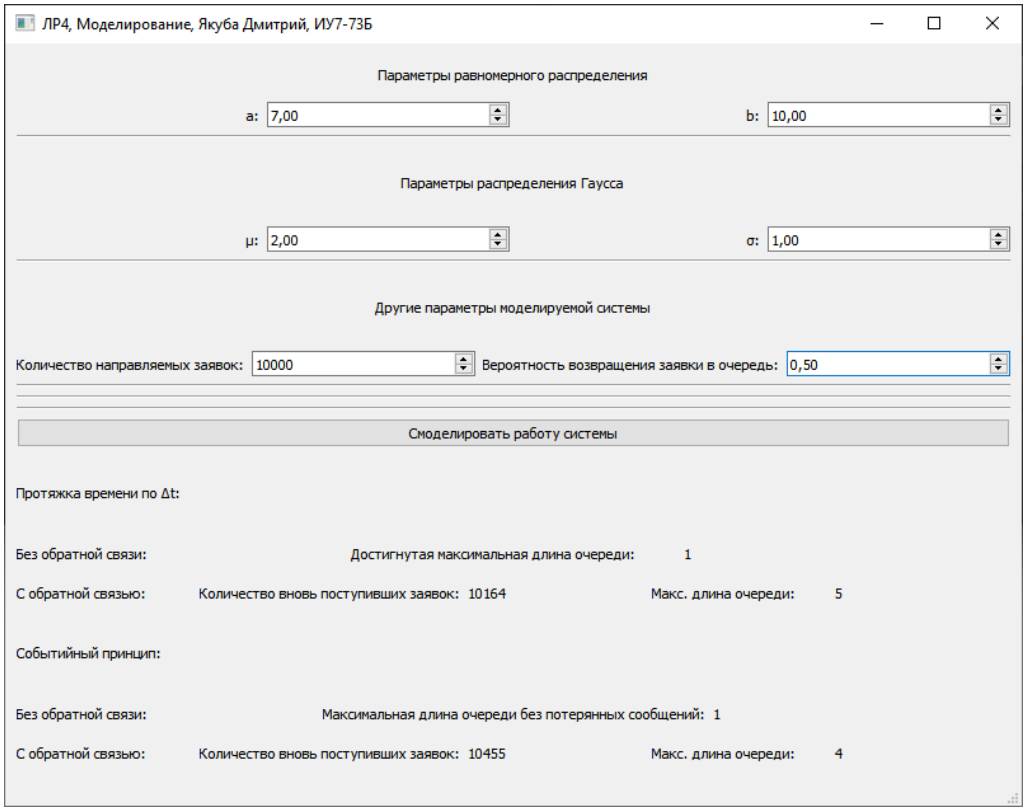

Рис. 3.5, Пример работы для системы с обратной связью с ОА, обрабатывающим запросы быстрее их генерации

4. Выводы

На рисунке 4.1 предоставлен один из результатов моделирования для системы, в которую поступает 1000 заявок, каждая из которых имеет 50% шанс вернуться в очередь.

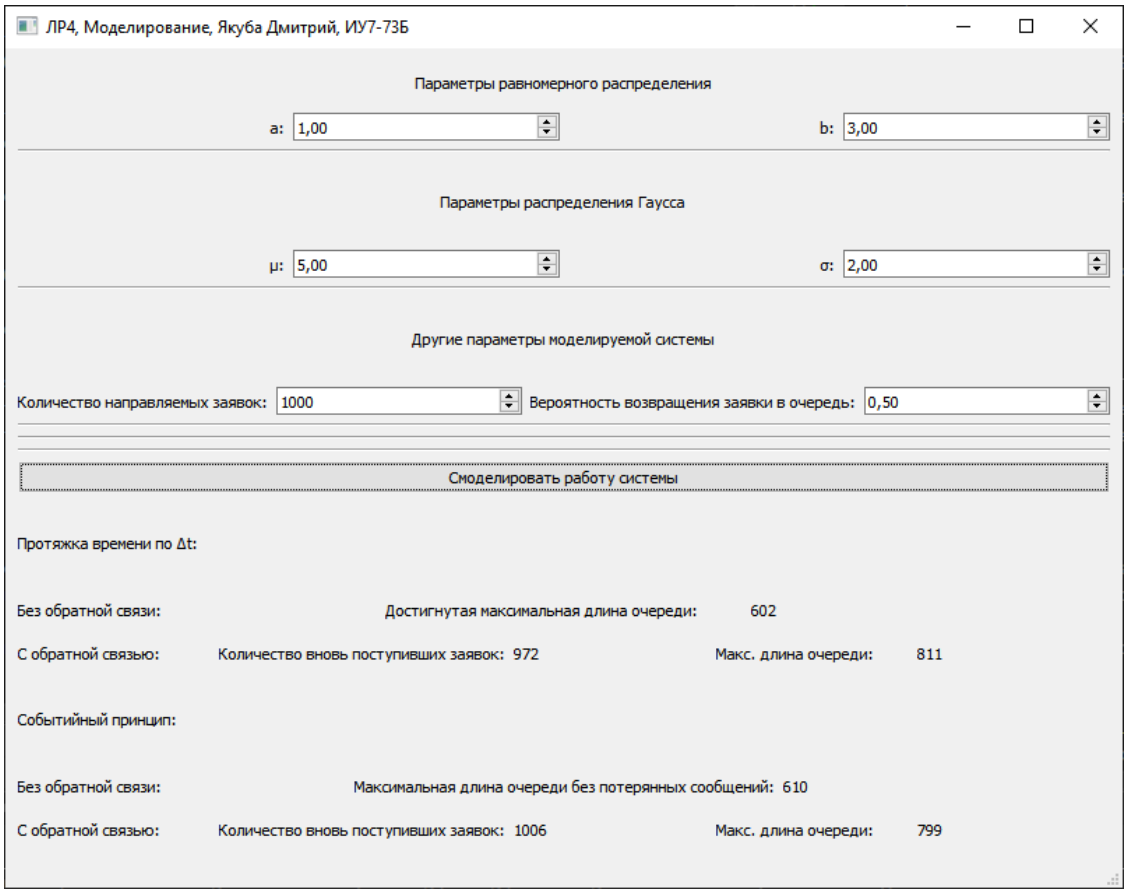

Рис. 4.1, Пример работы программы

Как видно из результатов: при моделировании вновь поступило около 1000 заявок. Это связано с фактом того, что каждая из вернувшихся в очередь заявок может вернуться в данную очередь вновь с той же заданной вероятностью в 0.5. Таким образом:

$$\frac{1000}{2} = 500; \frac{500}{2} = 250; \frac{250}{2} = 125; \frac{125}{2} \approx 63; \frac{63}{2} \approx 32; \frac{32}{2} = 16; \frac{16}{2} = 8;$$

$$\frac{8}{2} = 4; \frac{4}{2} = 2; \frac{2}{2} = 1;$$

$500 + 250 + 125 + 63 + 32 + 16 + 8 + 4 + 2 + 1 = 1001$, что и является ожидаемым значением количества вновь поступивших заявок.

Из рисунков 3.2-3.5 заметно, что результаты моделирования с использованием двух отличных способов протяжки модельного времени отличаются несущественно.

Основным недостатком протяжки времени по Δt является значительные затраты машинного времени на реализацию моделирования системы. При этом недостаточное малое Δt появляется опасность пропуска отдельных событий в системе, что исключает возможность получения адекватных результатов. В данной лабораторной работе $\Delta t = 1e^{-3}$.

Основным недостатком протяжки времени по событийному принципу является надобность в постоянном анализе списка будущих событий, что с большим количеством событий в системе может привести к большим затратам памяти и машинного времени.

5. Листинг

В данном разделе предоставлены используемые методы для построения требуемых графиков (используемый ЯП – C++).

```
double equalDistributionRandomValue(double aParameter, double bParameter)
{
    static std::mt19937 generator(std::random_device{}());
    std::uniform_real_distribution<double> distribution(aParameter,
bParameter);
    return distribution(generator);
}

double gaussDistributionRandomValue(double muParameter, double
sigmaParameter)
{
    static std::mt19937 generator(std::random_device{}());
    std::normal_distribution<double> distribution(muParameter,
sigmaParameter);
    return distribution(generator);
}

void Processor::processRequest()
{
    if (currentNumberOfRequestsInQueue == 0)
    {
        return;
    }

    currentNumberOfRequestsInQueue--;
    if (equalDistributionRandomValue(0, 1) < probabilityOfReturnToQueue)
    {
        numberOfReturnedRequests++;
        getRequest();
    }
}

void Processor::getRequest()
{
    currentNumberOfRequestsInQueue++;
```

```
    if (currentNumberOfRequestsInQueue > detectedMaxOfRequestsInQueue)
    {
        detectedMaxOfRequestsInQueue = currentNumberOfRequestsInQueue;
    }
}

double Processor::getNextTimeOfRequestProcessed()
{
    return gaussDistributionRandomValue(muParameter, sigmaParameter);
}

double RequestsGenerator::getNextTimeOfRequestGenerated()
{
    return equalDistributionRandomValue(aParameter, bParameter);
}

QPair<double, double> Simulator::simulateUsingDeltaTMethod(size_t
numberOfRequests)
{
    double timeOfGeneration =
requestGenerator.getNextTimeOfRequestGenerated();
    double timeOfProcessing =
        timeOfGeneration + processor.getNextTimeOfRequestProcessed();

    size_t numberOfSentRequests = 0;
    for (double currentTime = 0; numberOfSentRequests < numberOfRequests;
         currentTime += 1e-3)
    {
        while (timeOfGeneration <= currentTime)
        {
            numberOfSentRequests++;
            processor.getRequest();

            timeOfGeneration +=
requestGenerator.getNextTimeOfRequestGenerated();
        }

        while (timeOfProcessing <= currentTime)
        {
            processor.processRequest();

            (processor.currentNumberOfRequestsInQueue > 0)
                ? timeOfProcessing +=
processor.getNextTimeOfRequestProcessed()
                : timeOfProcessing = timeOfGeneration +

processor.getNextTimeOfRequestProcessed();
        }
    }

    while (processor.currentNumberOfRequestsInQueue > 0)
    {
        processor.processRequest();
    }

    return QPair<double, double>(
        processor.numberOfReturnedRequests,
processor.detectedMaxOfRequestsInQueue);
}

QPair<double, double> Simulator::simulateUsingEventMethod(size_t
numberOfRequests)
{
```

```
    double timeOfGeneration =
requestGenerator.getNextTimeOfRequestGenerated();
    double timeOfProcessing =
        timeOfGeneration + processor.getNextTimeOfRequestProcessed();

    size_t numberOfSentRequests = 0;
    while (numberOfSentRequests < numberOfRequests)
    {
        while (timeOfGeneration <= timeOfProcessing)
        {
            numberOfSentRequests++;
            processor.getRequest();

            timeOfGeneration +=
requestGenerator.getNextTimeOfRequestGenerated();
        }

        while (timeOfGeneration >= timeOfProcessing)
        {
            processor.processRequest();
            (processor.currentNumberOfRequestsInQueue > 0)
                ? timeOfProcessing +=
processor.getNextTimeOfRequestProcessed()
                : timeOfProcessing = timeOfGeneration +

processor.getNextTimeOfRequestProcessed();
        }
    }

    while (processor.currentNumberOfRequestsInQueue > 0)
    {
        processor.processRequest();
    }

    return QPair<double, double>(
        processor.numberOfReturnedRequests,
processor.detectedMaxOfRequestsInQueue);
}
```