# Лабораторная работа №5

## Дискретно-событийное моделирование

Цель работы:

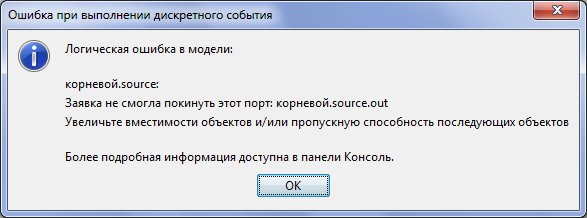
1. Изучить основные понятия теории систем массового обслуживания.
2. Изучить возможности AnyLogic для моделирования систем массового обслуживания.

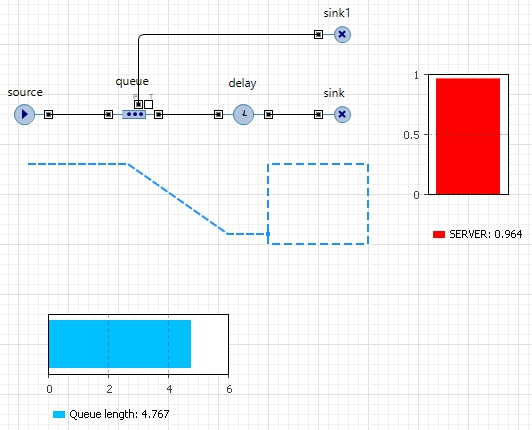
### Ход работы

**Задание:**

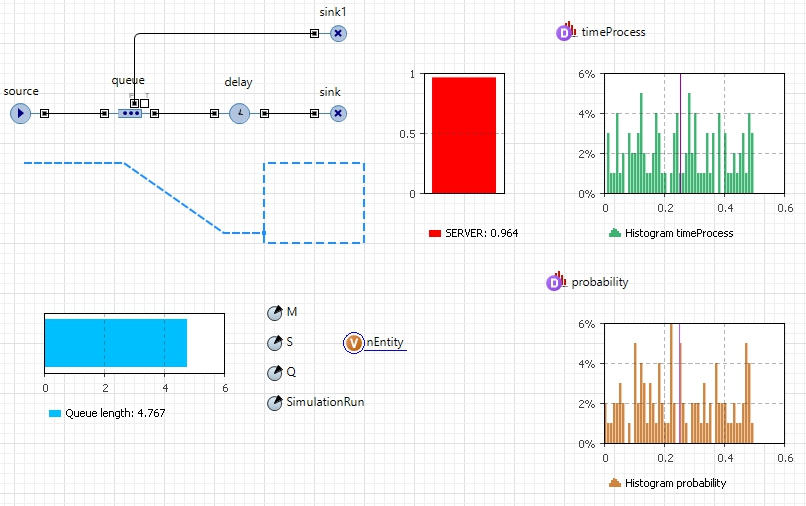
Сервер обрабатывает запросы, поступающие с автоматизированных рабочих мест с интервалами, распределёнными по показательному закону со средним значением 2 минуты. Время обработки сервером одного запроса распределено по закону со средним значением 3 минуты. Сервер имеет входной буфер ёмкостью 5 запросов. Построить имитационную модель для определения математического ожидания времени и вероятности обработки запросов.

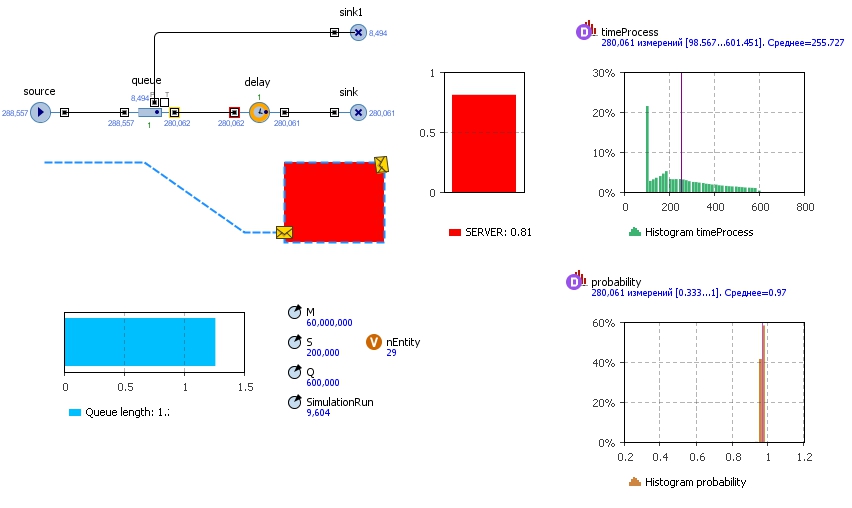
Сервер представляет собой одноканальную систему массового обслуживания с ограниченной очередью (надёжность системы считать абсолютной).

1. Создам модель с именем Server и единицами модельного времени — секундами.
2. Задам динамику процесса, создав диаграмму из блоков «Библиотеки моделирования процессов» (используя Source — генератор заявок, Queue — очередь заявок, Delay — задержка, Sink — конечная точка потока заявок).
3. Поскольку в условии сказано, что время поступления запросов на сервер 2 минуты, выставлю характеристику распределения объекта Source равной 1 / 120.
4. Задам длину очереди (объекта Queue) равной пяти и включу сбор статистики.
5. Объекту Delay задам характеристику распределения объекта равной 1 / 180 (согласно условию).
6. Выставив остановку в заданное время и фиксированное начальное число воспроизводимых прогонов запущу модель и увижу ошибку, связанную с переполнением очереди:  
   
7. Увеличу вместимость очереди до 15, чтобы на данном этапе ошибка не возникала. Позже решу эту проблему более верным способом.
8. Создам анимацию модели. Для этого добавлю на модель прямоугольный узел, который будет менять цвет заливки в зависимости от статуса сервера (красный — обрабатывает запрос, зелёный — нет).
9. Нарисую путь до прямоугольного узла в качестве фигуры анимации очереди. Запущу модель и проверю, что анимация работает.
10. Настрою сбор статистики использования ресурсов. Для этого создам две столбиковые диаграммы, отображающие статус сервера и среднюю длину очереди. Добавлю в качестве элементов данных функции delay.statsUtilization.mean() (statsUtilization — набор данных, занимающийся сбором статистики объекта delay) и queue.statsSize.mean() (statsSize выполняет аналогичную функцию).
11. Уточню модель согласно постановке задачи. Раньше у меня появлялась ошибка, связанная с переполнением очереди. Решать её увеличением очереди нельзя, поскольку это противоречит условию задачи. Решу проблему с помощью дополнительного объекта Sink, который будет уничтожать те заявки, которые не помещаются в очередь Queue. Для этого разрешу в настройках объекта Queue вытеснение. Вид программы на данном этапе (на следующей странице):



1. Настрою сбор статистических данных об обработанных запросах. Для этого создам нового агента с помощью элемента «Тип заявки». Назову его Inquiry и добавлю ему параметры timeIn, total и served типа double.
2. Изменю тип заявки объекта Source на Inquiry (а также у объектов Queue, Delay, Sink и Sink1), а действие при выходе задам равным entity.timeIn = time();, что позволит сохранять время создания заявки-запроса в параметре timeIn.
3. Создам гистограммы и данные гистограмм timeProcess и probability. У объекта Sink введу в поле действий при входе следующие коды:
   1. timeProcess.add(time() – entity.timeIn) — добавляет время обработки одного запроса в объект сбора данных гистограммы timeProcess.
   2. entity.served = sink.count() — записывает в served количество запросов, вошедших в блок Sink.
   3. entity.total = source.count() — записывает в total количество запросов вышедших из Source.
   4. probability.add(entity.served / entity.total) — добавляет долю обработанных запросов в объект сбор данных гистограммы probability.
4. Настрою гистограмму.
5. Поскольку изначально с целью упрощения процесса время обработки было принято распределённым по показательному закону со средним значением 3 минуты, а на самом деле время зависит от производительности сервера, то добавлю несколько новых параметров: M = 60 000 000, S = 200 000, Q = 600 000.
6. Создам параметр с именем SimulationRun и значением 9604.
7. Укажу объекту Delay время задержки normal(S, M) / Q, а к действие при входе у объекта Sink добавлю код: nEntity = round(sink.in.count() / SimulationRun), предварительно создав переменную nEntity.
8. Установлю конечное время модели равным 34 574 400. Результаты:





### Самостоятельная работа

Буду изменять среднее время поступления запросов (Time), ёмкость буфера (Capacity) и производительность сервера (Q), занося результаты в следующую таблицу:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Параметры моделирова-ния | Количество обработанных запросов | Вероятность обработки запросов | Среднее время обработки одного запроса | Средняя длина очереди запросов к серверу | Коэффициент использования сервера |
| Time = 120  Capacity = 5 | 280 064 | 0,97 | 255,726 | 1,261 | 0,81 |
| Time = 120  Capacity = 10 | 287 309 | 0,995 | 325,017 | 1,87 | 0,831 |
| Time = 40  Capacity = 5 | 345 742 | 0,4 | 554,961 | 4,55 | 1 |
| Time = 40  Capacity = 10 | 345 743 | 0,4 | 1 005,044 | 9,551 | 1 |
| Time = 40  Capacity = 10  Q = 1 000 000 | 576 217 | 0,667 | 591,15 | 8,852 | 1 |
| Time = 40  Capacity = 15  Q = 2 000 000 | 865 096 | 1 | 75,096 | 1,128 | 0,751 |

**Вывод:** чем больше производительность сервера, ёмкость буфера и время поступления запросов, тем больше вероятность обработки запроса. Если уменьшать эти параметры, то сервер перестаёт справляться с количеством поступающих запросов и потому вероятность обработки падает.