Модель системы массового обслуживания

Отчёт по лабораторной работе №11

Ибатулина Дарья Эдуардовна

Содержание

# 1 Введение

**Цель работы**

Реализовать модель в CPN tools.

**Задание**

* Реализовать в CPN Tools модель системы массового обслуживания M|M|1.
* Настроить мониторинг параметров моделируемой системы и нарисовать графики очереди [1].

# 2 Теоретическое введение

Модель системы массового обслуживания (СМО) **M|M|1** — одна из базовых моделей теории массового обслуживания, широко применяемая для анализа процессов обслуживания заявок в различных системах (телекоммуникации, вычислительные сети, производственные процессы и др.).

Обозначение **M|M|1** расшифровывается следующим образом: - Первая буква **M** (Markovian) означает, что время между поступлениями заявок в систему подчиняется экспоненциальному распределению (процесс поступления заявок — пуассоновский). - Вторая буква **M** указывает, что время обслуживания каждой заявки также экспоненциально распределено. - Цифра **1** означает, что в системе имеется один обслуживающий канал (один сервер) [2,3].

## 2.1 Основные характеристики модели M|M|1

* **Интенсивность потока заявок** — , среднее число заявок, поступающих в систему за единицу времени.
* **Интенсивность обслуживания** — , среднее число заявок, которые может обслужить сервер за единицу времени.
* **Коэффициент загрузки сервера** —
* , характеризует степень загруженности системы. Для устойчивой работы системы необходимо, чтобы выполнялось условие:

## 2.2 Основные параметры системы

* Среднее число заявок в системе:
* Среднее время пребывания заявки в системе:
* Среднее число заявок в очереди:
* Среднее время ожидания в очереди:

Модель **M|M|1** является классической и служит основой для построения более сложных моделей систем массового обслуживания [4–6].

# 3 Выполнение лабораторной работы

**Постановка задачи**

В систему поступает поток заявок двух типов, распределённый по пуассоновскому закону. Заявки поступают в очередь сервера на обработку. Дисциплина очереди - FIFO. Если сервер находится в режиме ожидания (нет заявок на сервере), то заявка поступает на обработку сервером.

Будем использовать три отдельных листа: на первом листе опишем граф системы (рис. [1](#fig:001)) и параллельно зададим декларации системы (рис. [2](#fig:002)), на втором — генератор заявок (рис. [3](#fig:003)), на третьем — сервер обработки заявок (рис. [4](#fig:004)).

Сеть имеет 2 позиции (очередь — Queue, обслуженные заявки — Completed) и два перехода (генерировать заявку — Arrivals, передать заявку на обработку сер- веру — Server). Переходы имеют сложную иерархическую структуру, задаваемую на отдельных листах модели (с помощью соответствующего инструмента меню — Hierarchy).

Между переходом Arrivals и позицией Queue, а также между позицией Queue и переходом Server установлена дуплексная связь. Между переходом Server и позицией Completed — односторонняя связь.

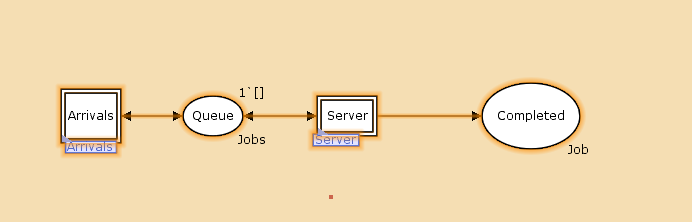


Figure 1: Граф сети системы обработки заявок в очереди

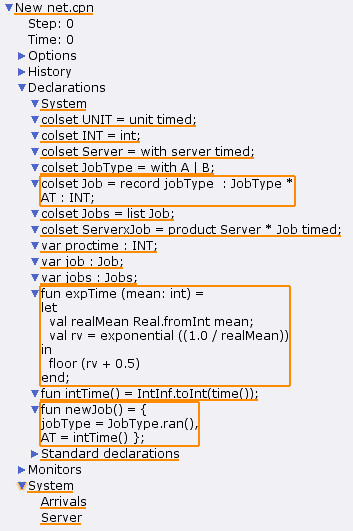


Figure 2: Декларации системы

Граф генератора заявок имеет 3 позиции (текущая заявка — Init, следующая заявка — Next, очередь — Queue из листа System) и 2 перехода (Init — определяет распределение поступления заявок по экспоненциальному закону с интенсивностью 100 заявок в единицу времени, Arrive — определяет поступление заявок в очередь).

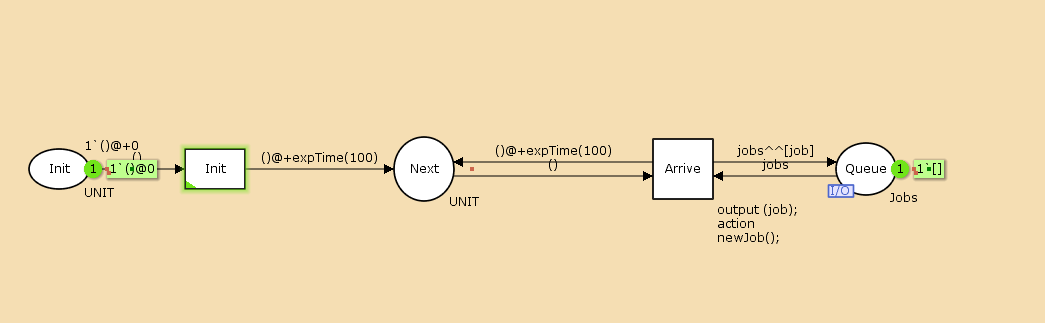


Figure 3: Граф генератора заявок системы

Граф процесса обработки заявок на сервере имеет 4 позиции (Busy — сервер занят, Idle — сервер в режиме ожидания, Queue и Completed из листа System) и 2 перехода (Start — начать обработку заявки, Stop — закончить обработку заявки).

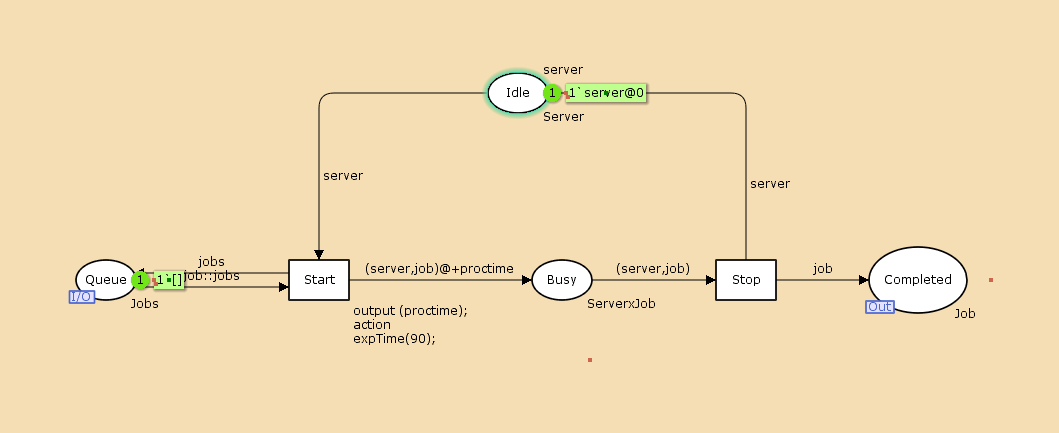


Figure 4: Граф процесса обработки заявок на сервере системы

Зададим декларации системы (рис. [[4](#fig:004)]).

Определим множества цветов системы (colorset):

* фишки типа UNIT определяют моменты времени;
* фишки типа INT определяют моменты поступления заявок в систему.
* фишки типа JobType определяют 2 типа заявок — A и B;
* кортеж Job имеет 2 поля: jobType определяет тип работы (соответственно имеет тип JobType, поле AT имеет тип INT и используется для хранения времени нахождения заявки в системе);
* фишки Jobs — список заявок;
* фишки типа ServerxJob — определяют состояние сервера, занятого обработкой заявок.

Переменные модели:

* proctime — определяет время обработки заявки;
* job — определяет тип заявки;
* jobs — определяет поступление заявок в очередь.

Определим функции системы:

* функция expTime описывает генерацию целочисленных значений через интервалы времени, распределённые по экспоненциальному закону;
* функция intTime преобразует текущее модельное время в целое число;
* функция newJob возвращает значение из набора Job — случайный выбор типа заявки (A или B).

Задание деклараций приведено выше (рис. [2](#fig:002))

Параметры модели на графах сети.

На листе System (рис. [1](#fig:001)):

* у позиции Queue множество цветов фишек — Jobs; начальная маркировка 1[] определяет, что изначально очередь пуста.
* у позиции Completed множество цветов фишек — Job.

На листе Arrivals (рис. [3](#fig:003)):

* у позиции Init: множество цветов фишек — UNIT; начальная маркировка 1``()@0 определяет, что поступление заявок в систему начинается с нулевого момента времени;
* у позиции Next: множество цветов фишек — UNIT;
* на дуге от позиции Init к переходу Init выражение () задаёт генерацию заявок;
* на дуге от переходов Init и Arrive к позиции Next выражение ()@+expTime(100) задаёт экспоненциальное распределение времени между поступлениями заявок;
* на дуге от позиции Next к переходу Arrive выражение () задаёт перемещение фишки;
* на дуге от перехода Arrive к позиции Queue выражение jobs^^[job] задает поступление заявки в очередь;
* на дуге от позиции Queue к переходу Arrive выражение jobs задаёт обратную связь.

На листе Server (рис. [4](#fig:004)):

* у позиции Busy: множество цветов фишек — Server, начальное значение маркировки — server определяет, что изначально на сервере нет заявок на обслуживание;
* у позиции Idle: множество цветов фишек — ServerxJob;
* переход Start имеет сегмент кода output (proctime); action expTime(90); определяющий, что время обслуживания заявки распределено по экспоненциальному закону со средним временем обработки в 90 единиц времени;
* на дуге от позиции Queue к переходу Start выражение job::jobs определяет, что сервер может начать обработку заявки, если в очереди есть хотя бы одна заявка;
* на дуге от перехода Start к позиции Busy выражение (server,job)@+proctime запускает функцию расчёта времени обработки заявки на сервере;
* на дуге от позиции Busy к переходу Stop выражение (server,job) говорит о завершении обработки заявки на сервере;
* на дуге от перехода Stop к позиции Completed выражение job показывает, что заявка считается обслуженной;
* выражение server на дугах от и к позиции Idle определяет изменение состояние сервера (обрабатывает заявки или ожидает);
* на дуге от перехода Start к позиции Queue выражение jobs задаёт обратную связь.

## 3.1 Запуск модели

Запускаем модель, добавляя, как сказано в руководстве, по 30 секунд после нажатия кнопки запуска (рис. [5](#fig:005), [6](#fig:006), [7](#fig:007)).

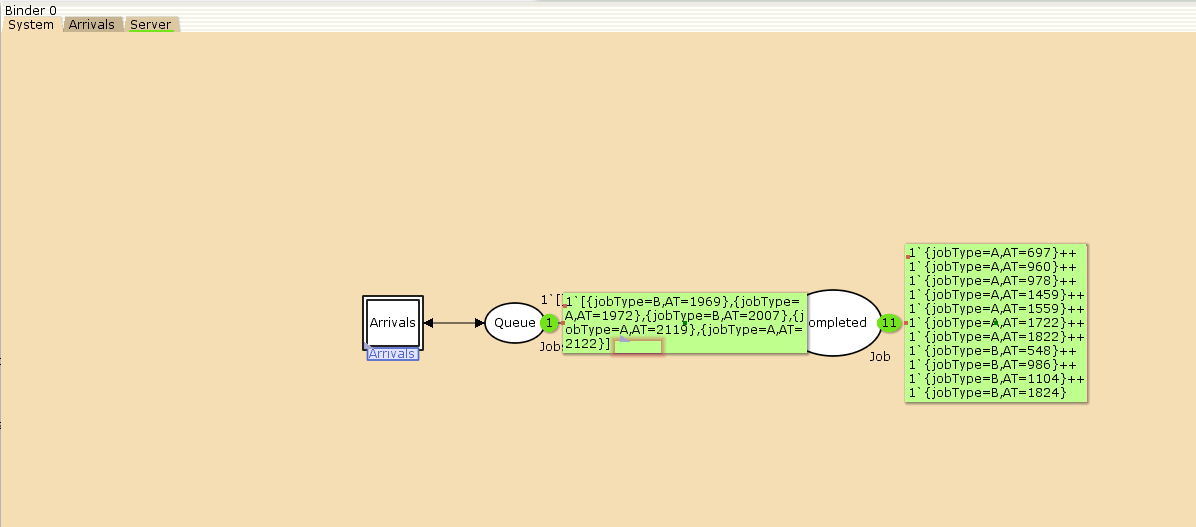


Figure 5: Граф сети системы обработки заявок в очереди

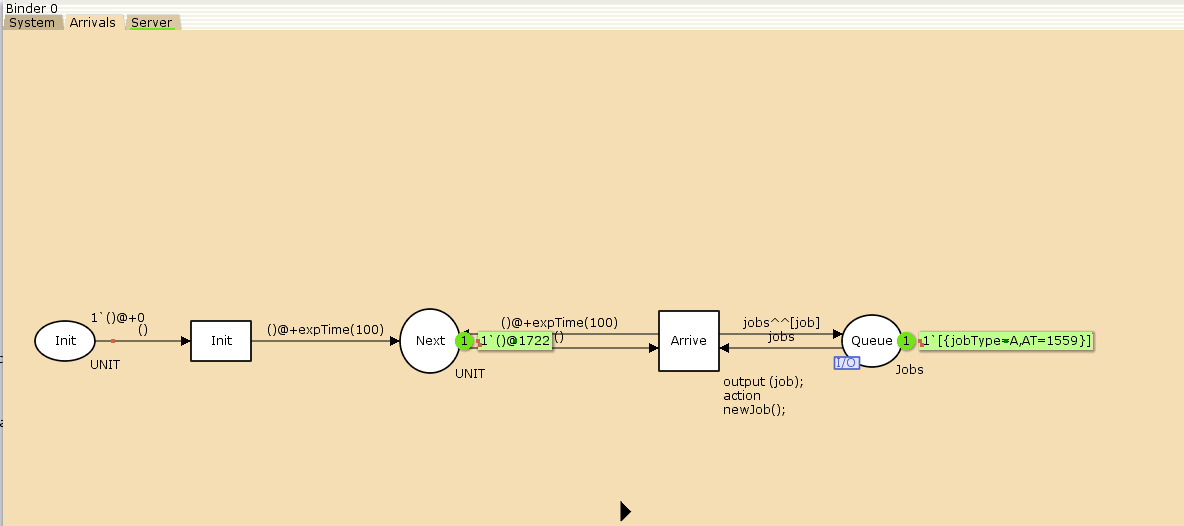


Figure 6: Граф генератора заявок системы

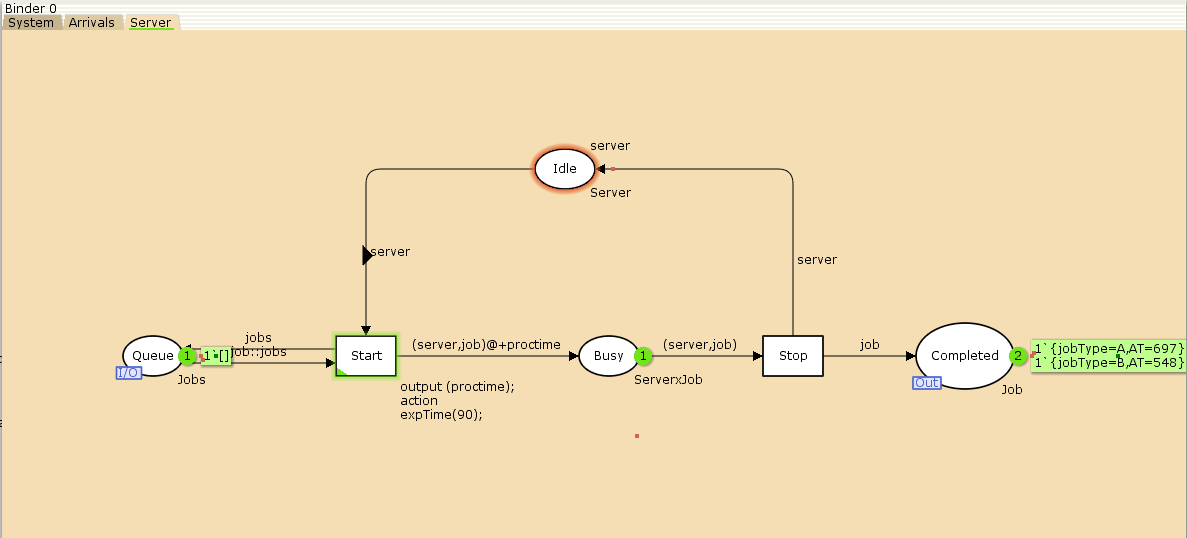


Figure 7: Граф процесса обработки заявок на сервере системы

## 3.2 Мониторинг параметров моделируемой системы

Потребуется палитра Monitoring. Выбираем Break Point (точка останова) и устанавливаем её на переход Start. После этого в разделе меню Monitor появится новый подраздел, который назовём Ostanovka. В этом подразделе необходимо внести изменения в функцию Predicate, которая будет выполняться при запуске монитора. Зададим число шагов, через которое будем останавливать мониторинг. Для этого true заменим на *Queue\_Delay.count()=200*.

В результате функция примет вид (рис. [8](#fig:008)):

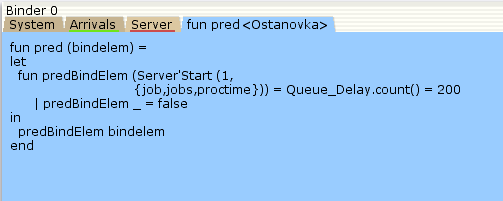


Figure 8: Функция Predicate монитора Ostanovka

Необходимо определить конструкцию Queue\_Delay.count(). С помощью палитры Monitoring выбираем Data Call и устанавливаем на переходе Start. Появившийся в меню монитор называем Queue Delay (без подчеркивания). Функция Observer выполняется тогда, когда функция предикатора выдаёт значение true. По умолчанию функция выдаёт 0 или унарный минус (~1), подчёркивание обозначает произвольный аргумент. Изменим её так, чтобы получить значение задержки в очереди. Для этого необходимо из текущего времени intTime() вычесть временную метку AT , означающую приход заявки в очередь.

В результате функция примет вид (рис. [9](#fig:009)):

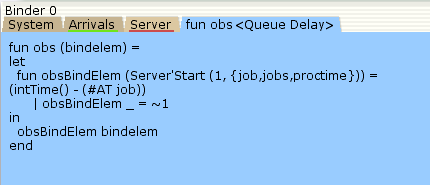


Figure 9: Функция Observer монитора Queue Delay

После запуска программы на выполнение в каталоге с кодом программы появится файл Queue\_Delay.log (рис. [10](#fig:010)), содержащий в первой колонке — значение задержки очереди, во второй — счётчик, в третьей — шаг, в четвёртой — время.

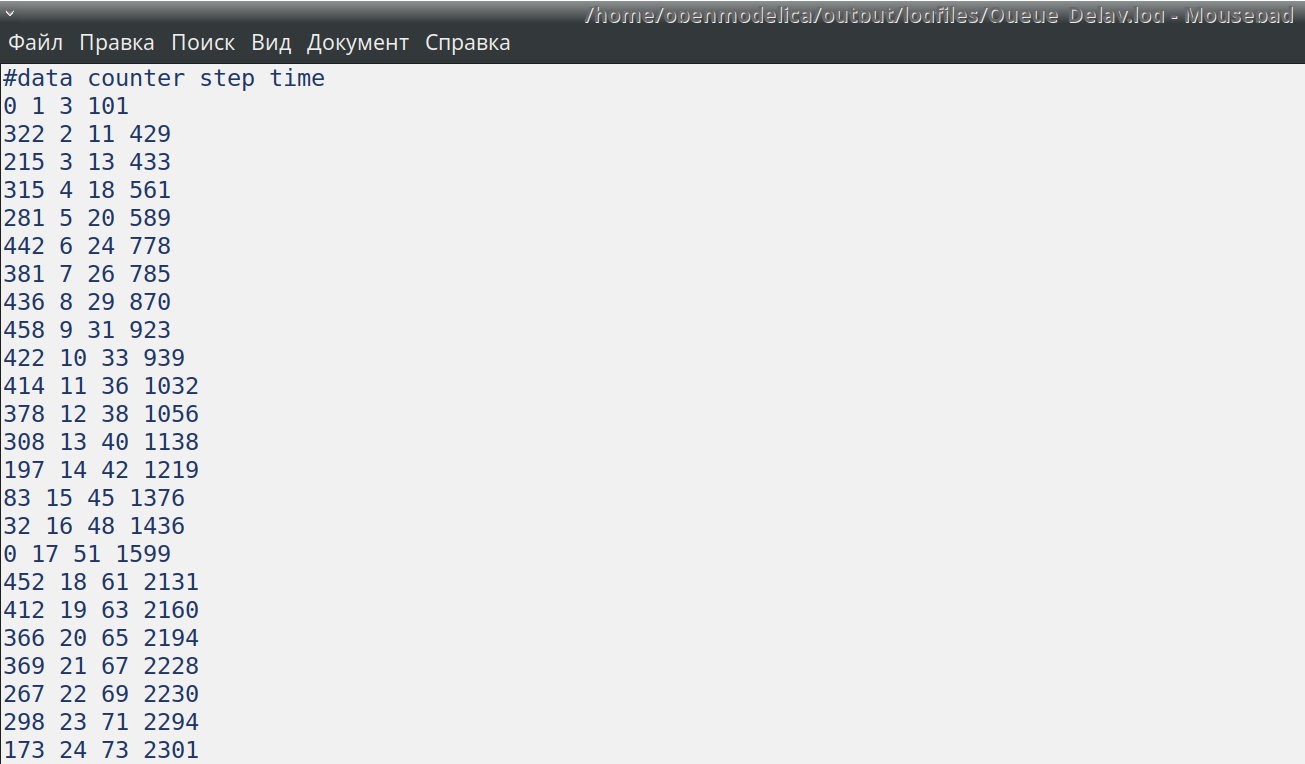


Figure 10: Файл Queue\_Delay.log

С помощью gnuplot можно построить график значений задержки в очереди (рис. [11](#fig:011)), выбрав по оси x время, а по оси y — значения задержки:

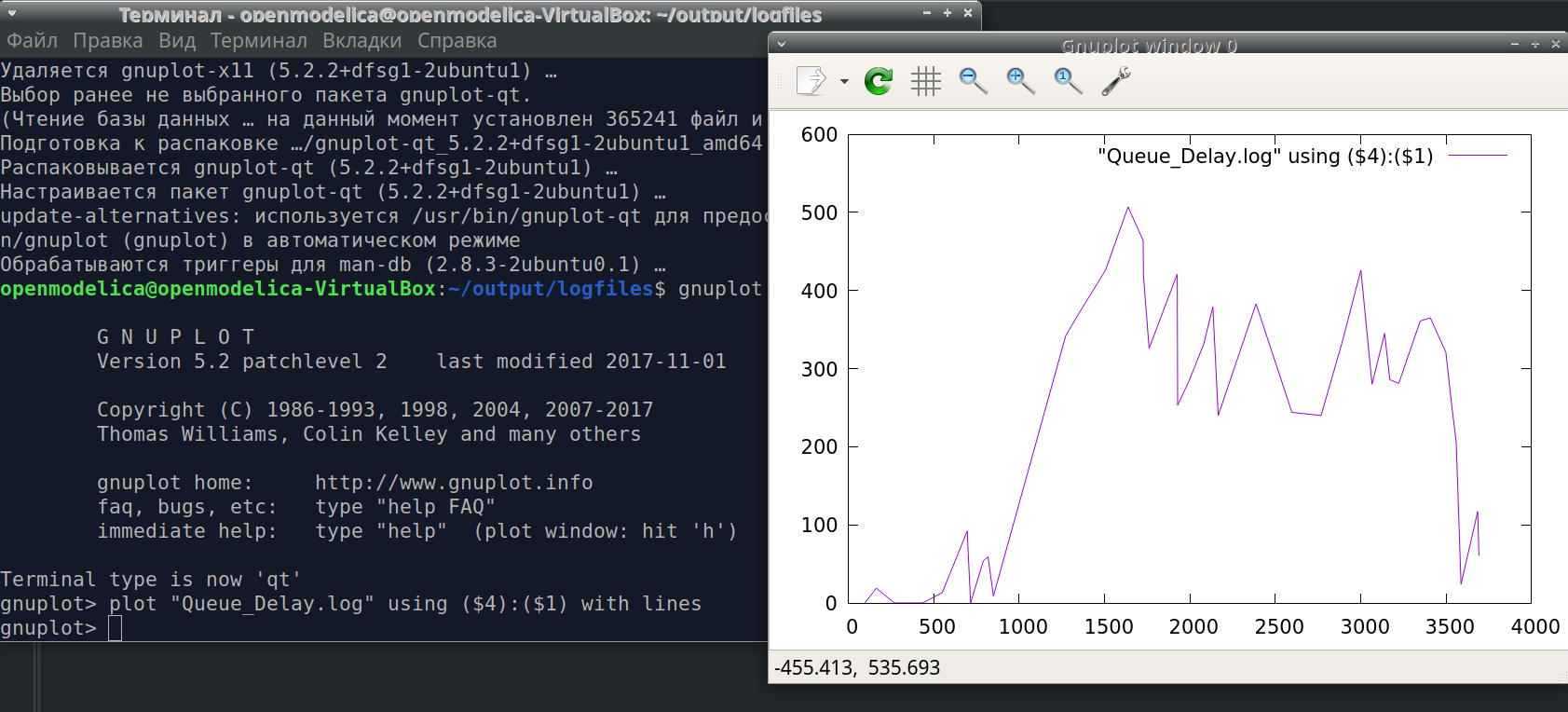


Figure 11: График изменения задержки в очереди

Посчитаем задержку в действительных значениях. С помощью палитры Monitoring выбираем Data Call и устанавливаем на переходе Start. Появившийся в меню монитор называем Queue Delay Real. Функцию Observer изменим следующим образом (рис. [12](#fig:012)):

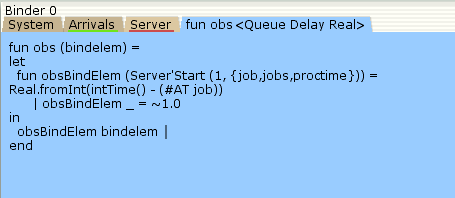


Figure 12: Функция Observer монитора Queue Delay Real

По сравнению с предыдущим описанием функции добавлено преобразование значения функции из целого в действительное, при этом obsBindElem \_ принимает значение ~1.0. После запуска программы на выполнение в каталоге с кодом программы появится файл Queue\_Delay\_Real.log с содержимым, аналогичным содержимому файла Queue\_Delay.log, но значения задержки имеют действительный тип (рис. [13](#fig:013)):

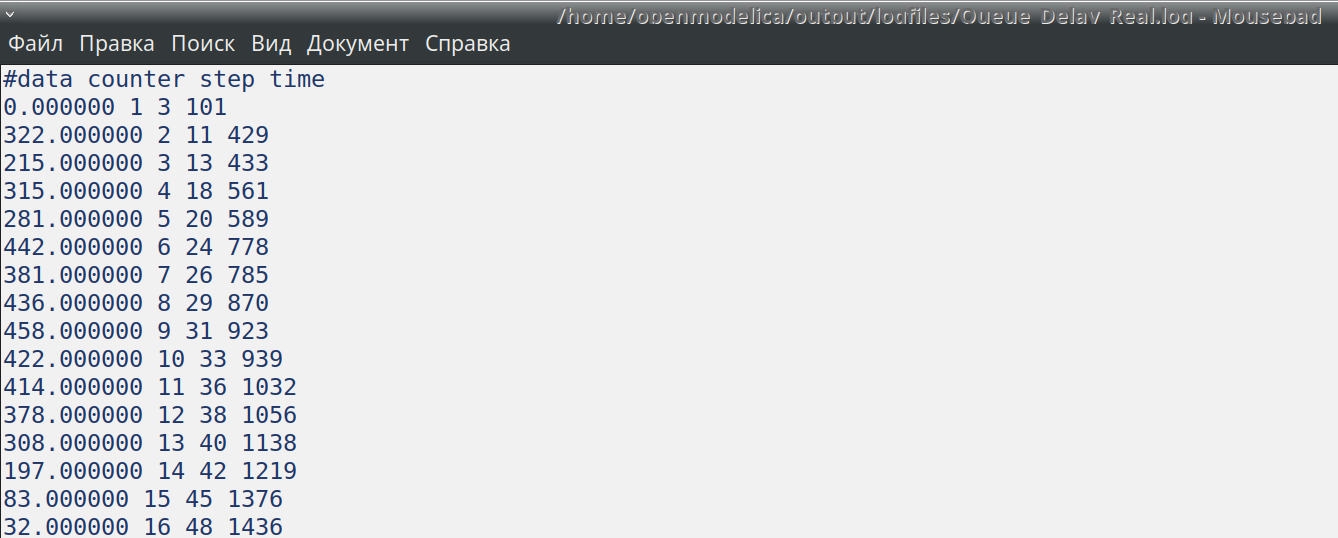


Figure 13: Содержимое Queue\_Delay\_Real.log

Посчитаем, сколько раз задержка превысила заданное значение. С помощью палитры Monitoring выбираем Data Call и устанавливаем на переходе Start. Монитор называем Long Delay Time. Функцию Observer изменим следующим образом (рис. [[14](#fig:014)]):

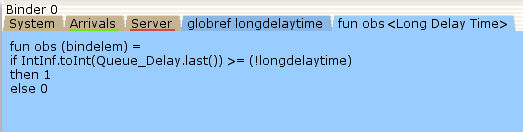


Figure 14: Функция Observer монитора Long Delay Time

При этом необходимо в декларациях задать глобальную переменную (в форме ссылки на число 200): longdelaytime (рис. [[15](#fig:015)]).

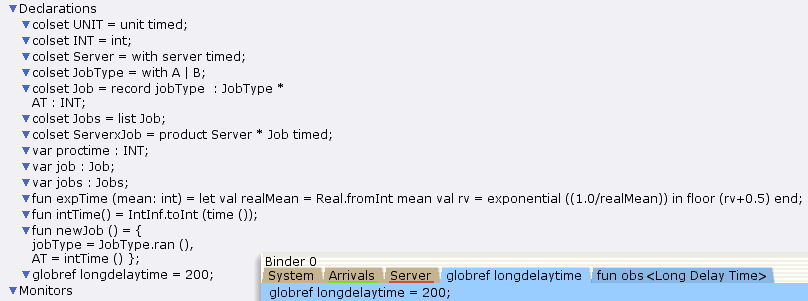


Figure 15: Определение longdelaytime в декларациях

После запуска программы на выполнение в каталоге с кодом программы появится файл Long\_Delay\_Time.log (рис. [[16](#fig:016)])

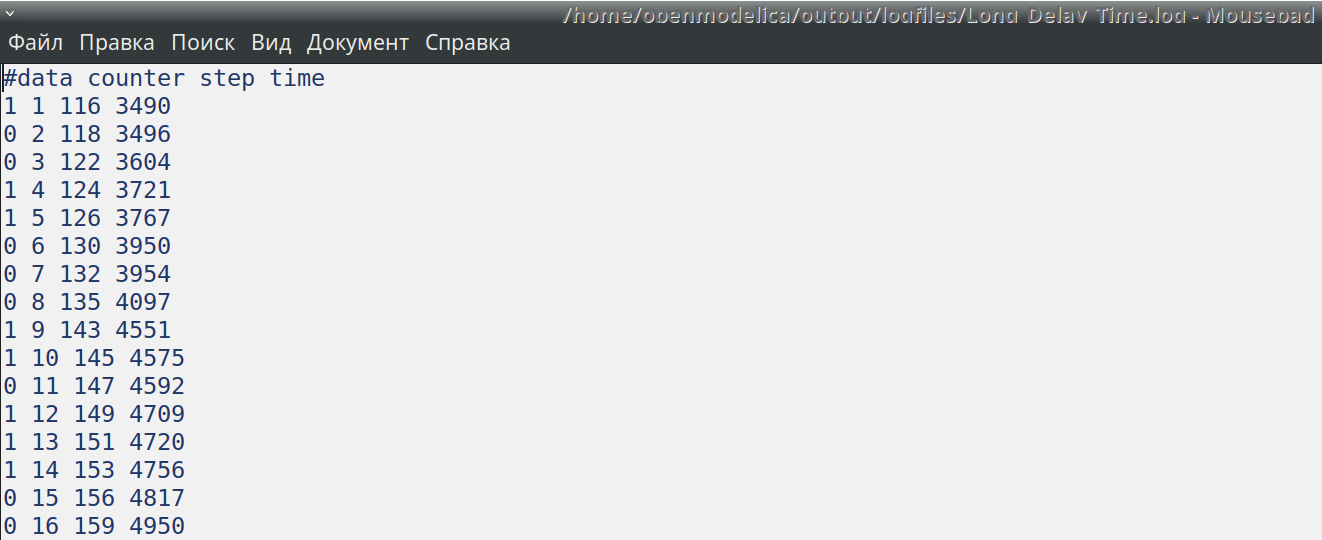


Figure 16: Содержимое Long\_Delay\_Time.log

С помощью gnuplot можно построить график (рис. [[17](#fig:017)]), демонстрирующий, в какие периоды времени значения задержки в очереди превышали заданное значение 200.

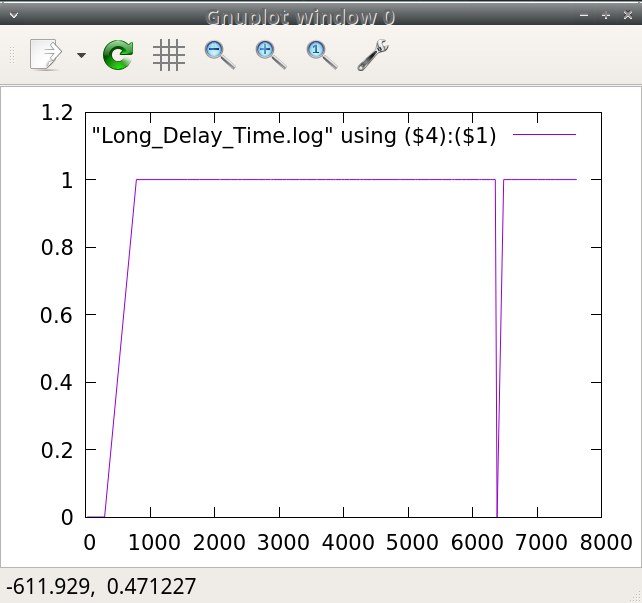


Figure 17: Периоды времени, когда значения задержки в очереди превышали заданное значение

# 4 Выводы

В процессе выполнения данной лабораторной работы я реализовала модель системы массового обслуживания в CPN Tools.

# Список литературы

1. Королькова А.В., Кулябов Д.С. Руководство к лабораторной работе №11. Модель системы массового обслуживания . 2025. С. 10.

2. Кузнецов В.В. Системы массового обслуживания. Юрайт, 2021.

3. Черушева Т.В., Зверовщикова Н.В. Теория массового обслуживания. Пенза: Изд-во ПГУ, 2021.

4. Плескунов М.А. Теория массового обслуживания. Уральский федеральный университет, 2022.

5. Романенко В.А. Системы и сети массового обслуживания. Самара: Издательство Самарского университета, 2021. С. 68.

6. Белый Е.К. Введение в теорию массового обслуживания. 2014.