

Модель системы массового обслуживания $M|M|1$

Отчёт по лабораторной работе №11

Ибатулина Дарья Эдуардовна

Содержание

1	Введение	4
2	Теоретическое введение	5
2.1	Основные характеристики модели $M M 1$	5
2.2	Основные параметры системы	6
3	Выполнение лабораторной работы	7
3.1	Запуск модели	13
3.2	Мониторинг параметров моделируемой системы	14
4	Выводы	21
	Список литературы	22

Список иллюстраций

3.1	Граф сети системы обработки заявок в очереди	8
3.2	Декларации системы	9
3.3	Граф генератора заявок системы	10
3.4	Граф процесса обработки заявок на сервере системы	10
3.5	Граф сети системы обработки заявок в очереди	13
3.6	Граф генератора заявок системы	14
3.7	Граф процесса обработки заявок на сервере системы	14
3.8	Функция Predicate монитора Ostanovka	15
3.9	Функция Observer монитора Queue Delay	15
3.10	Файл Queue_Delay.log	16
3.11	График изменения задержки в очереди	16
3.12	Функция Observer монитора Queue Delay Real	17
3.13	Содержимое Queue_Delay_Real.log	17
3.14	Функция Observer монитора Long Delay Time	18
3.15	Определение longdelaytime в декларациях	18
3.16	Содержимое Long_Delay_Time.log	19
3.17	Периоды времени, когда значения задержки в очереди превышали заданное значение	20

1 Введение

Цель работы

Реализовать модель $M|M|1$ 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** — λ , среднее число заявок, поступающих в систему за единицу времени.
- **Интенсивность обслуживания** — μ , среднее число заявок, которые может обслужить сервер за единицу времени.
- **Коэффициент загрузки сервера** —

$$\rho = \frac{\lambda}{\mu}$$

, характеризует степень загруженности системы. Для устойчивой работы

системы необходимо, чтобы выполнялось условие:

$$\rho < 1.$$

2.2 Основные параметры системы

- Среднее число заявок в системе:

$$L = \frac{\rho}{1 - \rho}$$

- Среднее время пребывания заявки в системе:

$$W = \frac{1}{\mu - \lambda}$$

- Среднее число заявок в очереди:

$$L_q = \frac{\rho^2}{1 - \rho}$$

- Среднее время ожидания в очереди:

$$W_q = \frac{\rho}{\mu - \lambda}$$

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. 3.1) и параллельно зададим декларации системы (рис. 3.2), на втором — генератор заявок (рис. 3.3), на третьем — сервер обработки заявок (рис. 3.4).

Сеть имеет 2 позиции (очередь — Queue, обслуженные заявки — Completed) и два перехода (генерировать заявку — Arrivals, передать заявку на обработку серверу — 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.

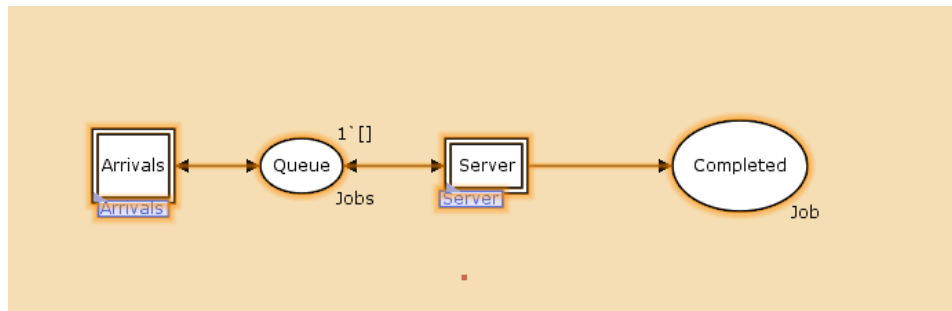


Рис. 3.1: Граф сети системы обработки заявок в очереди


```

▼ New net.cpn
  Step: 0
  Time: 0
  ▶ Options
  ▶ History
  ▼ Declarations
    ▼ System
    ▼ colset UNIT = unit timed;
    ▼ colset INT = int;
    ▼ colset Server = with server timed;
    ▼ colset JobType = with A | B;
    ▼ colset Job = record jobType : JobType *
      AT : INT;
    ▼ colset Jobs = list Job;
    ▼ colset ServerxJob = product Server * Job timed;
    ▼ var proctime : INT;
    ▼ var job : Job;
    ▼ var jobs : Jobs;
    ▼ fun expTime (mean: int) =
      let
        val realMean Real.fromInt mean;
        val rv = exponential ((1.0 / realMean))
      in
        floor (rv + 0.5)
      end;
    ▼ fun intTime() = IntInf.toInt(time());
    ▼ fun newJob() = {
      jobType = JobType.ran(),
      AT = intTime() };
    ▶ Standard declarations
  ▶ Monitors
  ▼ System
    Arrivals
    Server

```

Рис. 3.2: 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).

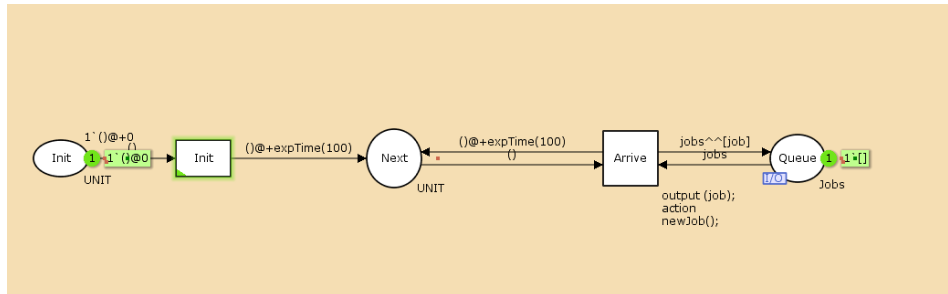


Рис. 3.3: Граф генератора заявок системы

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).

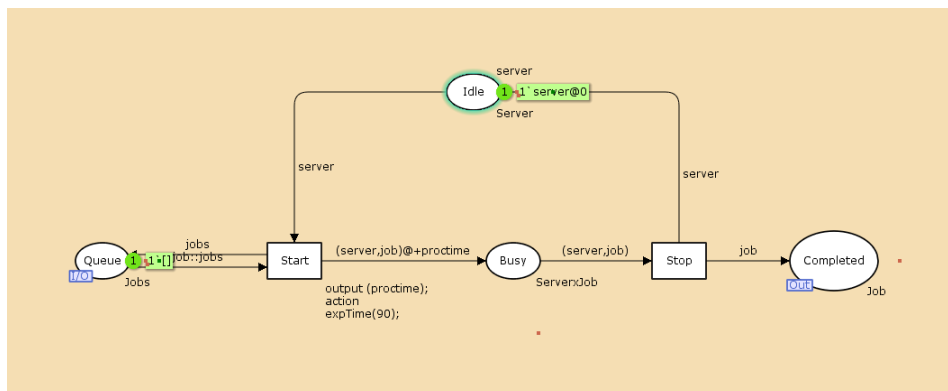


Рис. 3.4: Граф процесса обработки заявок на сервере системы

Зададим декларации системы (рис. [3.4]).

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 — А и В;

- 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. 3.2)

Параметры модели на графах сети.

На листе System (рис. 3.1):

- 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. 3.3):

- 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. 3.4):

- 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 состояния 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. 3.5, 3.6, 3.7).

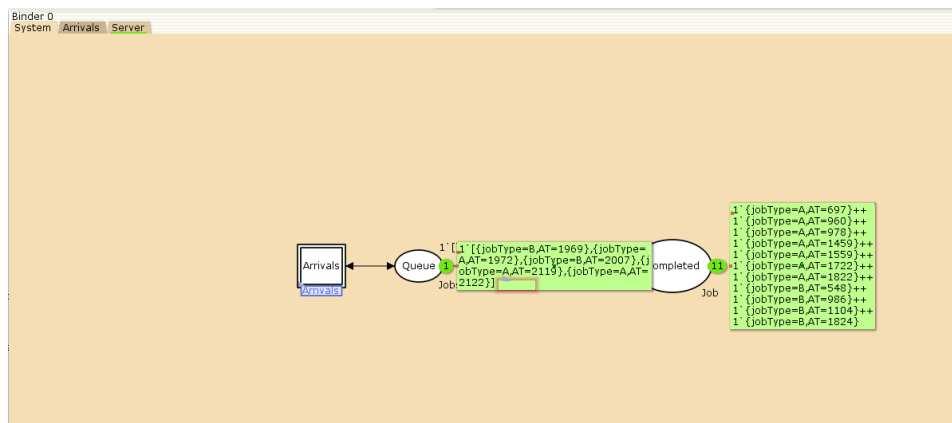


Рис. 3.5: Граф сети системы обработки заявок в очереди

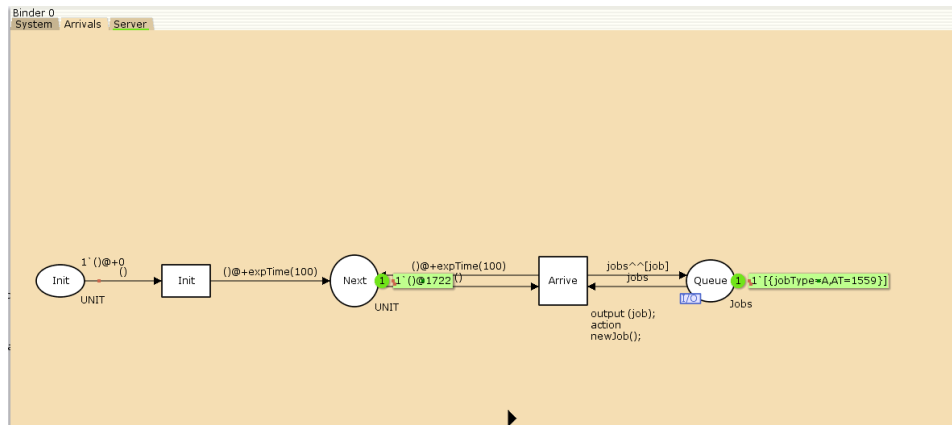


Рис. 3.6: Граф генератора заявок системы

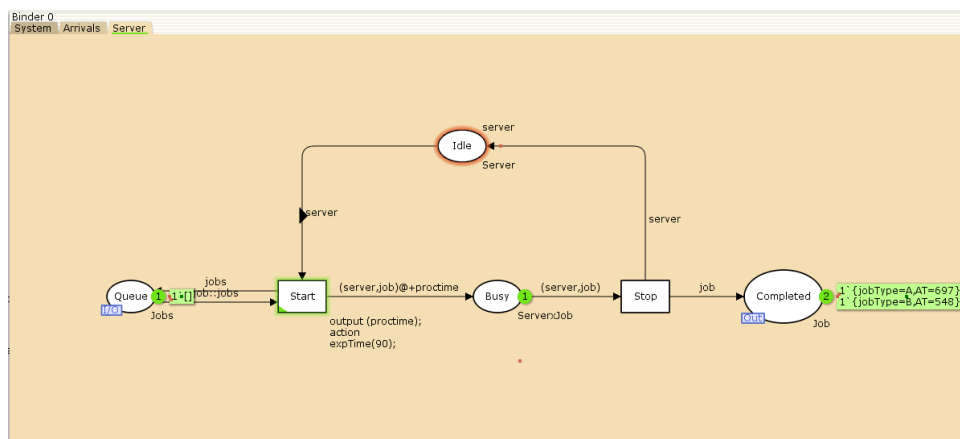
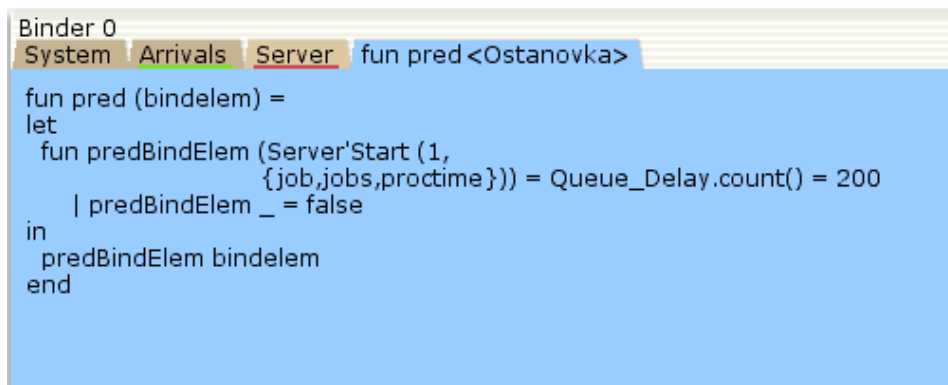


Рис. 3.7: Граф процесса обработки заявок на сервере системы

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. 3.8):

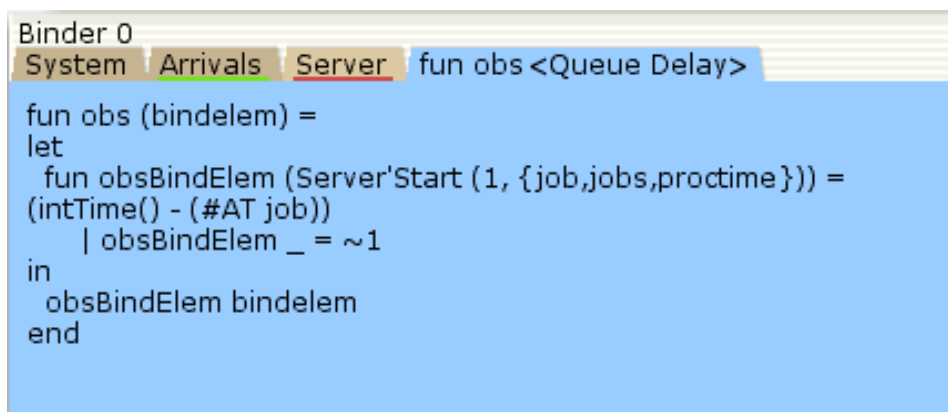


```
Binder 0
System Arrivals Server fun pred<Ostanovka>
fun pred (bindelem) =
let
  fun predBindElem (Server'Start (1,
                                {job,jobs,proctime})) = Queue_Delay.count() = 200
  | predBindElem _ = false
in
  predBindElem bindelem
end
```

Рис. 3.8: 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.

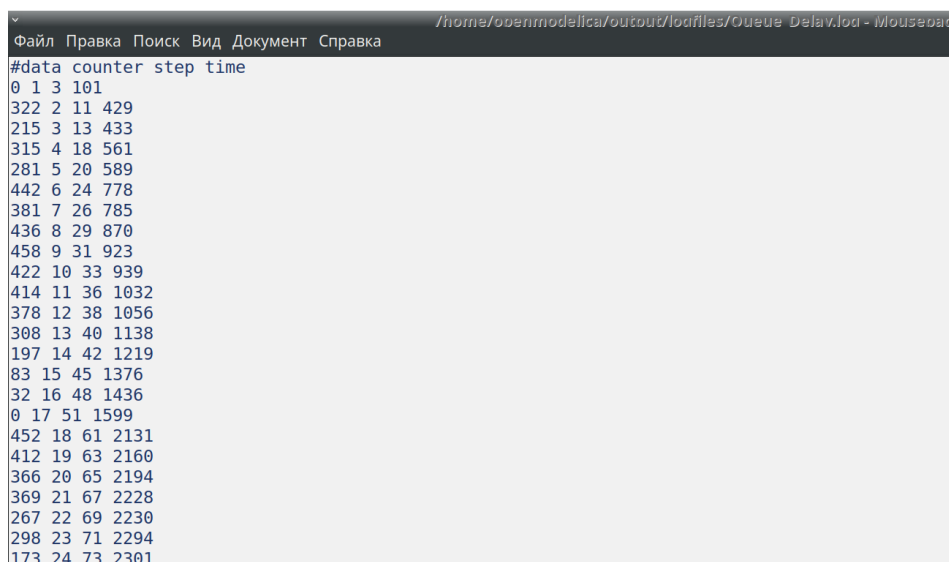
В результате функция примет вид (рис. 3.9):



```
Binder 0
System Arrivals Server fun obs<Queue Delay>
fun obs (bindelem) =
let
  fun obsBindElem (Server'Start (1, {job,jobs,proctime})) =
    (intTime() - (#AT job))
  | obsBindElem _ = ~1
in
  obsBindElem bindelem
end
```

Рис. 3.9: Функция Observer монитора Queue Delay

После запуска программы на выполнение в каталоге с кодом программы появится файл Queue_Delay.log (рис. 3.10), содержащий в первой колонке — значение задержки очереди, во второй — счётчик, в третьей — шаг, в четвёртой — время.



data	counter	step	time
0	1	3	101
322	2	11	429
215	3	13	433
315	4	18	561
281	5	20	589
442	6	24	778
381	7	26	785
436	8	29	870
458	9	31	923
422	10	33	939
414	11	36	1032
378	12	38	1056
308	13	40	1138
197	14	42	1219
83	15	45	1376
32	16	48	1436
0	17	51	1599
452	18	61	2131
412	19	63	2160
366	20	65	2194
369	21	67	2228
267	22	69	2230
298	23	71	2294
173	24	73	2301

Рис. 3.10: Файл Queue_Delay.log

С помощью gnuplot можно построить график значений задержки в очереди (рис. 3.11), выбрав по оси x время, а по оси y — значения задержки:

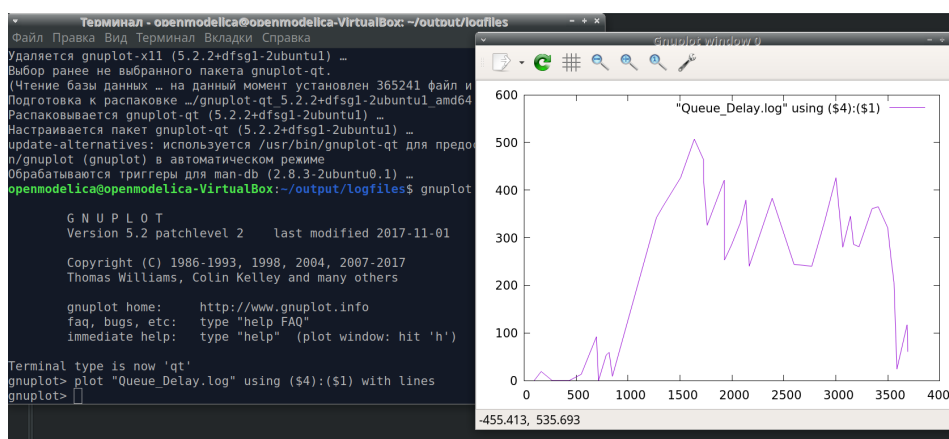
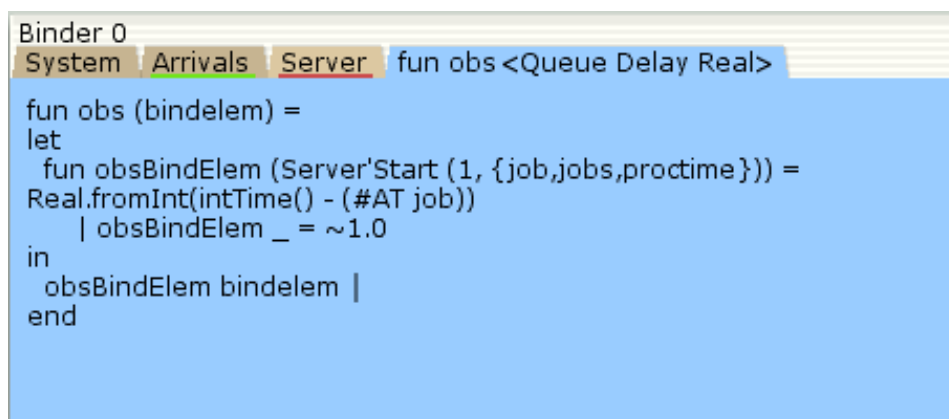


Рис. 3.11: График изменения задержки в очереди

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. 3.12):

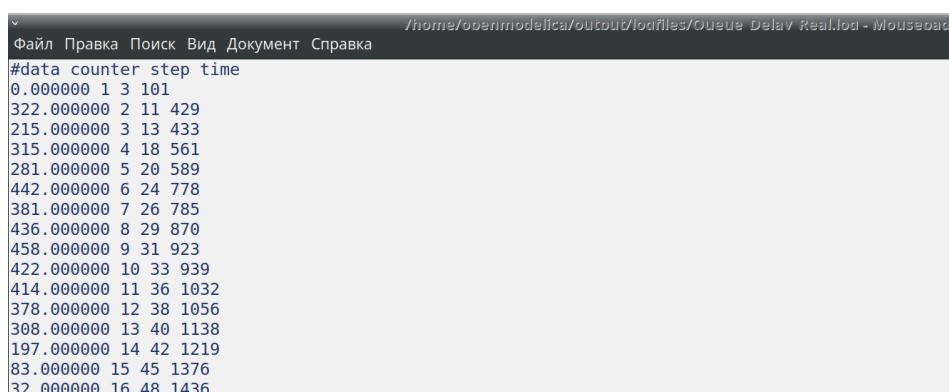


```

Binder 0
System Arrivals Server fun obs <Queue Delay Real>
fun obs (bindelem) =
let
  fun obsBindElem (Server'Start (1, {job,jobs,proctime})) =
    Real.fromInt(intTime() - (#AT job))
    | obsBindElem _ = ~1.0
in
  obsBindElem bindelem |
end
  
```

Рис. 3.12: 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. 3.13):



```

/home/ooenmodelica/output/lofiles/Queue_Delay_Real.log - Mousepad
Файл Правка Поиск Вид Документ Справка
#data counter step time
0.000000 1 3 101
322.000000 2 11 429
215.000000 3 13 433
315.000000 4 18 561
281.000000 5 20 589
442.000000 6 24 778
381.000000 7 26 785
436.000000 8 29 870
458.000000 9 31 923
422.000000 10 33 939
414.000000 11 36 1032
378.000000 12 38 1056
308.000000 13 40 1138
197.000000 14 42 1219
83.000000 15 45 1376
32.000000 16 48 1436
  
```

Рис. 3.13: 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. [3.14]):

```

Binder 0
System Arrivals Server globref longdelaytime fun obs <Long Delay Time>
fun obs (bindelem) =
if IntInf.toInt(Queue_Delay.last()) >= (!longdelaytime)
then 1
else 0
  
```

Рис. 3.14: Функция Observer монитора Long Delay Time

При этом необходимо в декларациях задать глобальную переменную (в форме ссылки на число 200): longdelaytime (рис. [3.15]).

```

▼ Declarations
  ▼ colset UNIT = unit timed;
  ▼ colset INT = int;
  ▼ colset Server = with server timed;
  ▼ colset JobType = with A | B;
  ▼ colset Job = record jobType : JobType *
    AT : INT;
  ▼ colset Jobs = list Job;
  ▼ colset ServerxJob = product Server * Job timed;
  ▼ var proctime : INT;
  ▼ var job : Job;
  ▼ var jobs : Jobs;
  ▼ fun expTime (mean: int) = let val realMean = Real.fromInt mean val rv = exponential ((1.0/realMean)) in floor (rv+0.5) end;
  ▼ fun intTime() = IntInf.toInt (time ());
  ▼ fun newJob () = {
    jobType = JobType.ran (),
    AT = intTime () };
  ▼ globref longdelaytime = 200;
▼ Monitors
  Binder 0
  System Arrivals Server globref longdelaytime fun obs <Long Delay Time>
  globref longdelaytime = 200;
  
```

Рис. 3.15: Определение longdelaytime в декларациях

После запуска программы на выполнение в каталоге с кодом программы появится файл Long_Delay_Time.log (рис. [3.16])

```

#data counter step time
1 1 116 3490
0 2 118 3496
0 3 122 3604
1 4 124 3721
1 5 126 3767
0 6 130 3950
0 7 132 3954
0 8 135 4097
1 9 143 4551
1 10 145 4575
0 11 147 4592
1 12 149 4709
1 13 151 4720
1 14 153 4756
0 15 156 4817
0 16 159 4950

```

Рис. 3.16: Содержимое Long_Delay_Time.log

С помощью gnuplot можно построить график (рис. [3.17]), демонстрирующий, в какие периоды времени значения задержки в очереди превышали заданное значение 200.

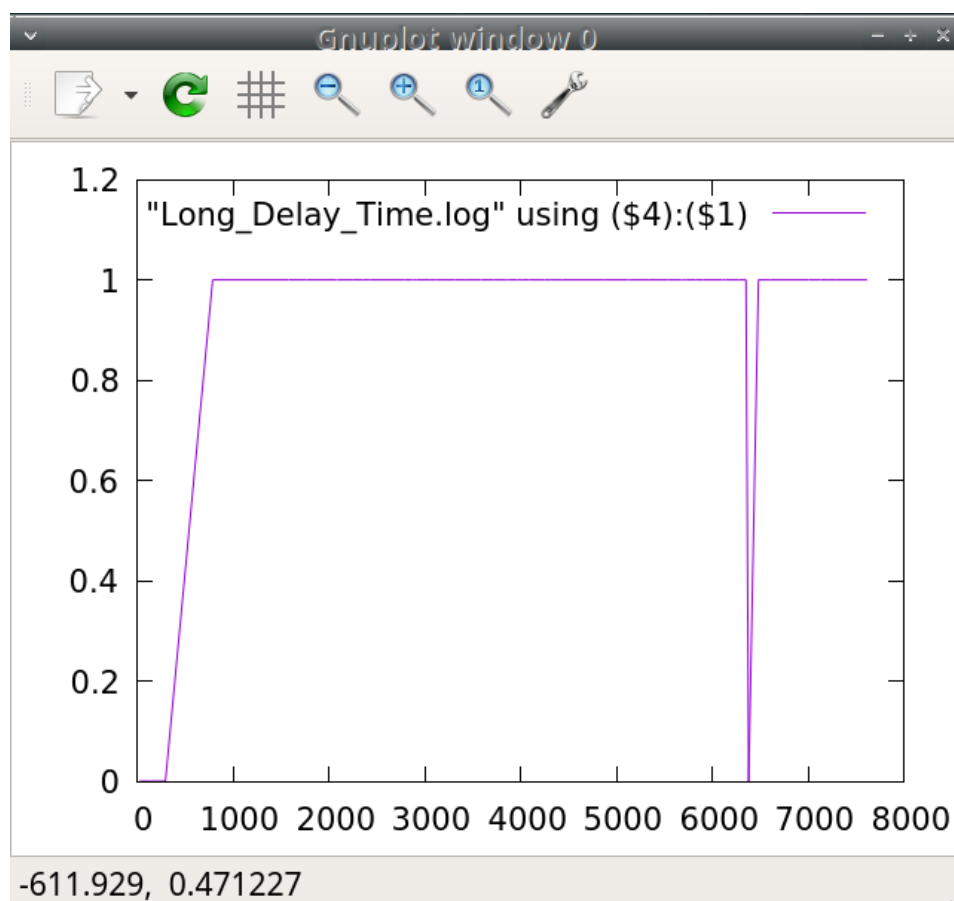


Рис. 3.17: Периоды времени, когда значения задержки в очереди превышали заданное значение

4 Выводы

В процессе выполнения данной лабораторной работы я реализовала модель системы массового обслуживания $M|M|1$ в CPN Tools.

Список литературы

1. Королькова А.В., Кулябов Д.С. Руководство к лабораторной работе №11. Модель системы массового обслуживания $M|M|1$. 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.