

# **Лабораторная работа №11**

**Модель системы массового обслуживания М/М/1/∞**

Кадров Виктор Максимович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
3.1	Постановка задачи . . . . .	6
3.2	Построение модели с помощью CPNTools . . . . .	6
3.3	Мониторинг параметров моделируемой системы . . . . .	13
<b>4</b>	<b>Выводы</b>	<b>20</b>
	<b>Список литературы</b>	<b>21</b>

## Список иллюстраций

3.1	Граф сети системы обработки заявок в очереди . . . . .	7
3.2	Граф генератора заявок системы . . . . .	7
3.3	Граф процесса обработки заявок на сервере системы . . . . .	8
3.4	Заданная декларация системы . . . . .	10
3.5	Параметры элементов основного графа системы обработки заявок в очереди . . . . .	11
3.6	Параметры элементов генератора заявок системы . . . . .	12
3.7	Параметры элементов обработчика заявок системы . . . . .	13
3.8	Функция Predicate монитора Ostanovka . . . . .	14
3.9	Функция Observer монитора Queue Delay . . . . .	14
3.10	Файл Queue_Delay.log . . . . .	15
3.11	График изменения задержки в очереди . . . . .	16
3.12	Функция Observer монитора Queue Delay Real . . . . .	16
3.13	Содержимое Queue_Delay_Real.log . . . . .	17
3.14	Функция Observer монитора Long Delay Time . . . . .	17
3.15	Определение longdelaytime в декларациях . . . . .	18
3.16	Содержимое Long_Delay_Time.log . . . . .	18
3.17	Периоды времени, когда значения задержки в очереди превышали заданное значение . . . . .	19

# 1 Цель работы

Реализовать модель  $M|M|1|\infty$  в *CPN tools*[1].

## 2 Задание

- Реализовать в CPN Tools модель системы массового обслуживания  $M|M|1|\infty$ .
- Настроить мониторинг параметров моделируемой системы и нарисовать графики изменения задержки в очереди[2].

## **3 Выполнение лабораторной работы**

### **3.1 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.

### **3.2 Построение модели с помощью CPNTools**

Будем использовать три отдельных листа: на первом листе опишем граф системы (рис. 3.1), на втором — генератор заявок (рис. 3.2), на третьем — сервер обработки заявок (рис. 3.3).

Сеть имеет 2 позиции (очередь — Queue, обслуженные заявки — Complited) 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 Complited — односторонняя связь.

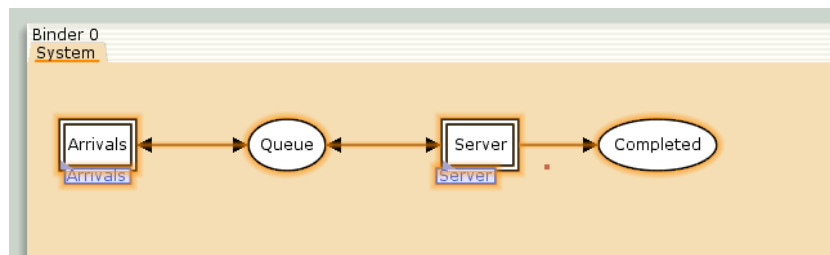


Рис. 3.1: Граф сети системы обработки заявок в очереди

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).

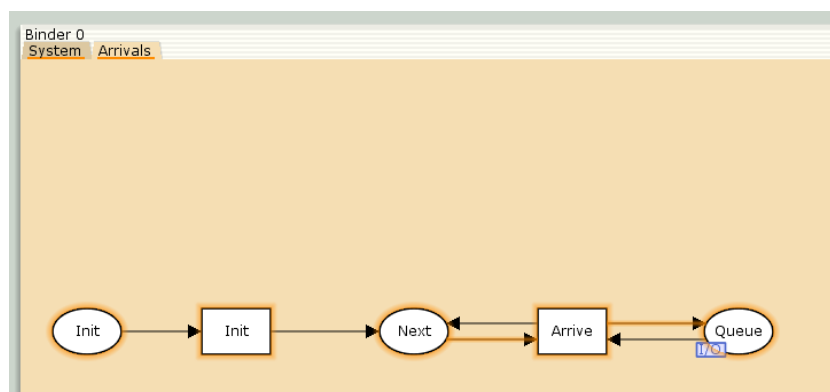


Рис. 3.2: Граф генератора заявок системы

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).

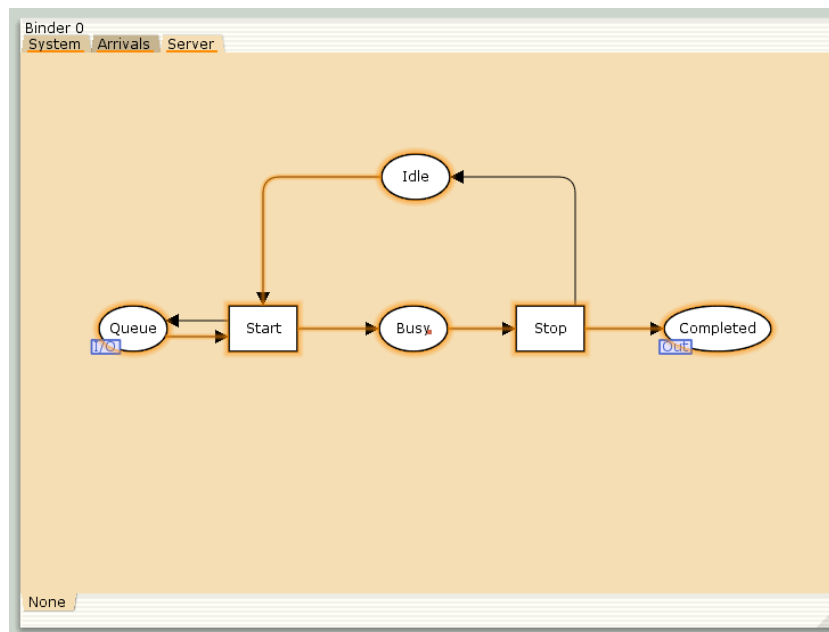


Рис. 3.3: Граф процесса обработки заявок на сервере системы

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 (А или В).

```

▼ lab11.cpn
  Step: 50
  Time: 2013
  ► Options
  ► History
  ▼ Declarations
    ▼ globref longdelaytime = 200;
    ▼ SYSTEM
      ▼ Standard declarations
        ▼ colset BOOL = bool;
        ▼ colset STRING = string;
        ▼ colset INT = int;
        ▼ colset UNIT = unit timed;
        ▼ colset Server = with server timed;
        ▼ colset JobType = with A | B;
        ► colset Job
        ▼ colset Jobs = list Job;
        ► colset ServerxJob
        ▼ var proctime : INT;
        ▼ var job : Job;
        ► var jobs
        ► fun expTime
        ▼ fun intTime() = IntInf.toInt (time());
        ▼ fun newJob() = {
          jobType = JobType.ran(),
          AT = intTime() };
      ► Monitors
      ▼ SMO
        Arrivals
        Server

```

Рис. 3.4: Заданная декларация системы

Зададим параметры модели на графах сети.

На листе System (рис. 3.5):

- у позиции Queue множество цветов фишек — Jobs; начальная маркировка

- $1[]$  определяет, что изначально очередь пуста.
- у позиции Completed множество цветов фишек — Job.

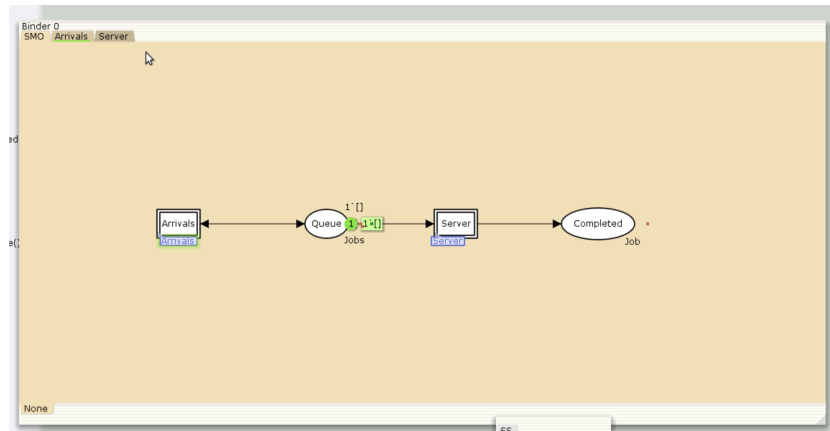


Рис. 3.5: Параметры элементов основного графа системы обработки заявок в очереди

На листе Arrivals (рис. 3.6):

- 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.

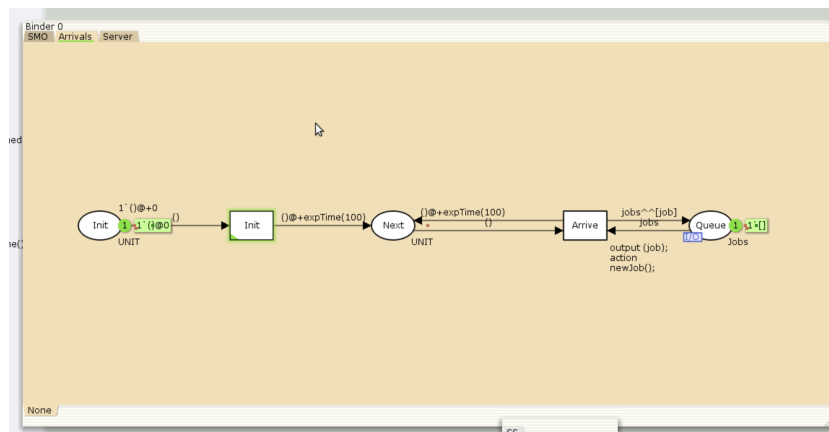


Рис. 3.6: Параметры элементов генератора заявок системы

На листе Server (рис. 3.7):

- у позиции Busy: множество цветов фишек — Server, начальное значение маркировки —  $1' \text{server}@0$  определяет, что изначально на сервере нет заявок на обслуживание;
- у позиции Idle: множество цветов фишек —  $\text{Server} \times \text{Job}$ ;
- 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.

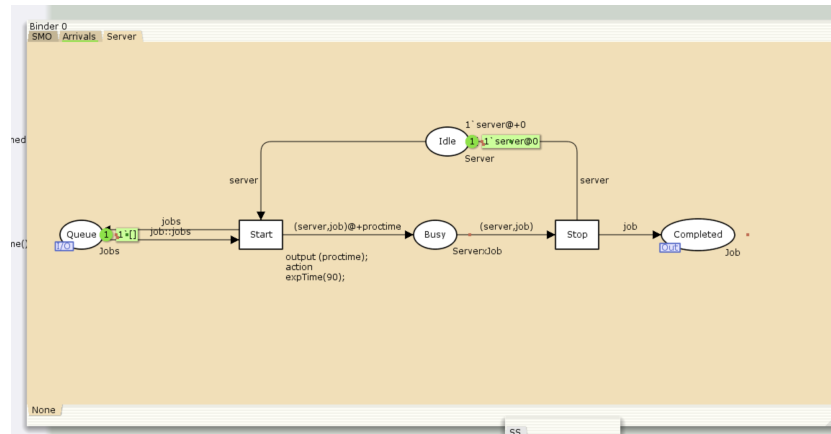
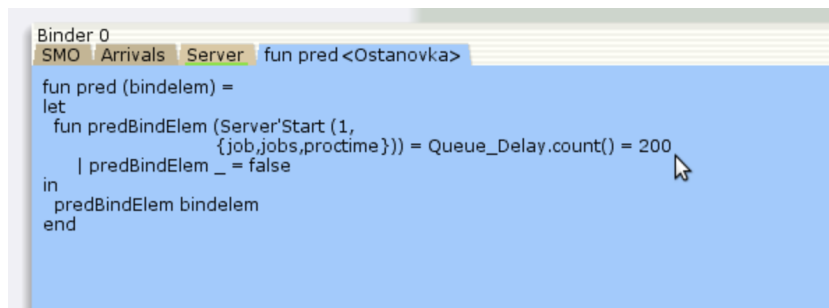


Рис. 3.7: Параметры элементов обработчика заявок системы

### 3.3 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. Изначально, когда функция начинает работать, она возвращает значение true, в противном случае — false. В теле функции вызывается процедура predBindElem, которую определяем в предварительных декларациях. 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):



```

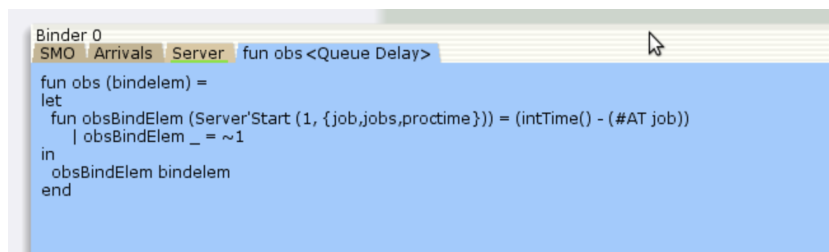
Binder 0
SMO Arrivals Server fun pred<Ostanovka>
fun pred (bindelem) =
let
  fun predBindElem (Server'Start (1, {job,jobs,proctime})) = Queue_Delay.count() = 200
  | predBindElem _ = false
in
  predBindElem bindelem
end

```

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):



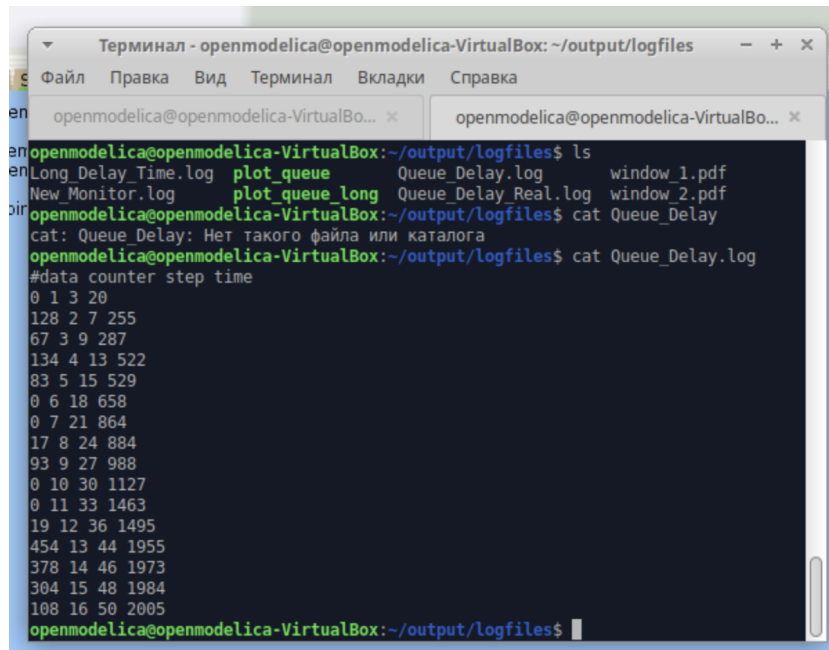
```

Binder 0
SMO 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.

A screenshot of a terminal window titled "Терминал - openmodelica@openmodelica-VirtualBox: ~/output/logfiles". The terminal shows the following commands and output:

```
openmodelica@openmodelica-VirtualBox:~/output/logfiles$ ls
Long_Delay_Time.log  plot_queue  Queue_Delay.log  window_1.pdf
New_Monitor.log     plot_queue_long  Queue_Delay_Real.log  window_2.pdf
openmodelica@openmodelica-VirtualBox:~/output/logfiles$ cat Queue_Delay
cat: Queue_Delay: Нет такого файла или каталога
openmodelica@openmodelica-VirtualBox:~/output/logfiles$ cat Queue_Delay.log
#data counter step time
0 1 3 20
128 2 7 255
67 3 9 287
134 4 13 522
83 5 15 529
0 6 18 658
0 7 21 864
17 8 24 884
93 9 27 988
0 10 30 1127
0 11 33 1463
19 12 36 1495
454 13 44 1955
378 14 46 1973
304 15 48 1984
108 16 50 2005
openmodelica@openmodelica-VirtualBox:~/output/logfiles$
```

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:

```
#!/usr/bin/gnuplot -persist
# задаём текстовую кодировку,
# тип терминала, тип и размер шрифта

set encoding utf8
set term pdfcairo font "Arial,9"

# задаём выходной файл графика
set out 'window_1.pdf'
plot "Queue_Delay.log" using ($4):($1) with lines
```

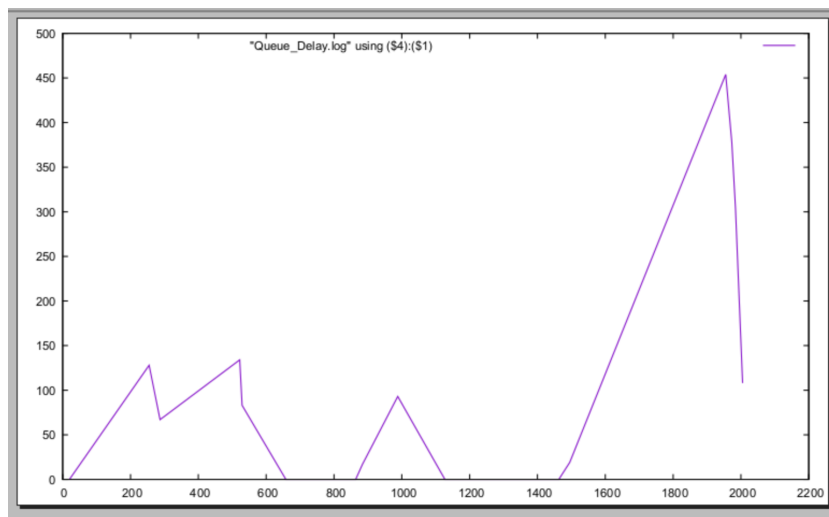


Рис. 3.11: График изменения задержки в очереди

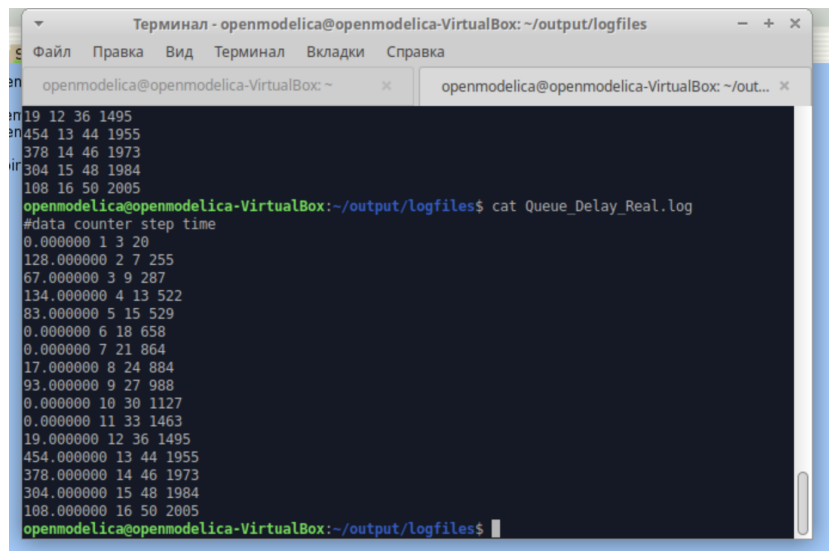
Посчитаем задержку в действительных значениях. С помощью палитры Monitoring выбираем Data Call и устанавливаем на переходе Start. Появившийся в меню монитор называем Queue Delay Real. Функцию Observer изменим следующим образом(рис. 3.12):

```
Binder 0
SMO 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):

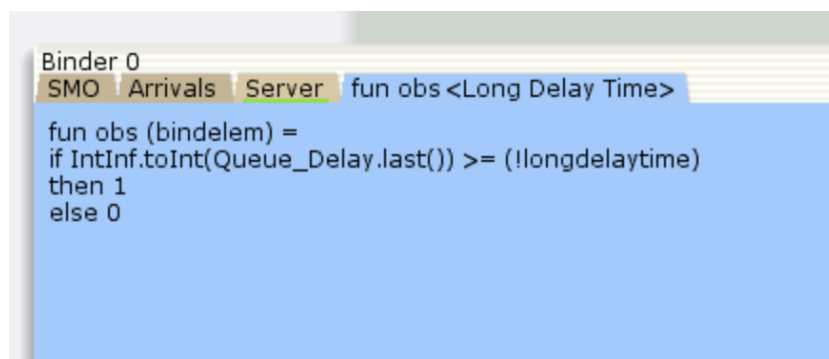




```
Терминал - openmodelica@openmodelica-VirtualBox: ~/output/logfiles
Файл  Правка  Вид  Терминал  Вкладки  Справка
openmodelica@openmodelica-VirtualBox: ~
openmodelica@openmodelica-VirtualBox: ~/out...
19 12 36 1495
454 13 44 1955
378 14 46 1973
304 15 48 1984
108 16 50 2005
openmodelica@openmodelica-VirtualBox:~/output/logfiles$ cat Queue_Delay_Real.log
#data counter step time
0.000000 1 3 20
128.000000 2 7 255
67.000000 3 9 287
134.000000 4 13 522
83.000000 5 15 529
0.000000 6 18 658
0.000000 7 21 864
17.000000 8 24 884
93.000000 9 27 988
0.000000 10 30 1127
0.000000 11 33 1463
19.000000 12 36 1495
454.000000 13 44 1955
378.000000 14 46 1973
304.000000 15 48 1984
108.000000 16 50 2005
openmodelica@openmodelica-VirtualBox:~/output/logfiles$
```

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
SMO Arrivals Server 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

Если значение монитора Queue Delay превысит некоторое заданное значение, то функция выдаст 1, в противном случае — 0. Восклицательный знак означает разыменованное значение ссылки.

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

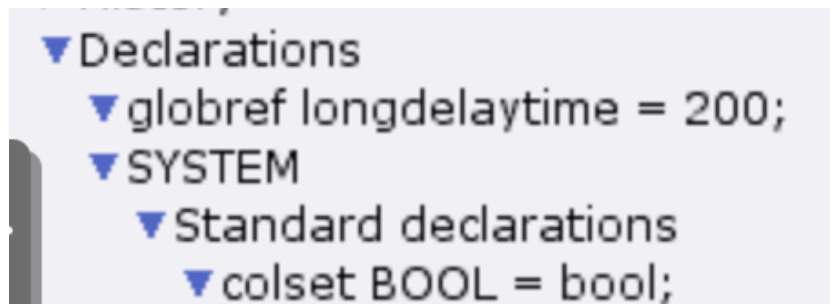


Рис. 3.15: Определение longdelaytime в декларациях

После запуска программы на выполнение в каталоге с кодом программы появится файл Long\_Delay\_Time.log (рис. 3.16)

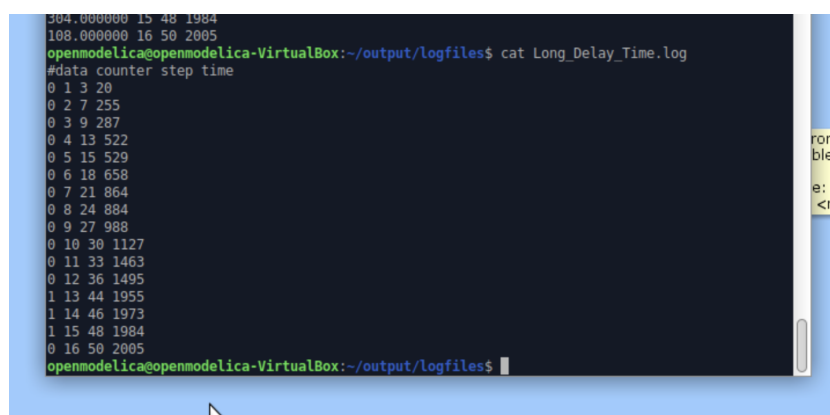


Рис. 3.16: Содержимое Long\_Delay\_Time.log

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

```
#!/usr/bin/gnuplot -persist
# задаём текстовую кодировку,
# тип терминала, тип и размер шрифта
```

```
set encoding utf8
set term pdfcairo font "Arial,9"
```

```
# задаём выходной файл графика  
set out 'window_2.pdf'  
plot [0:][0:1.2] "Long_Delay_Time.log" using ($4):($1) with lines
```

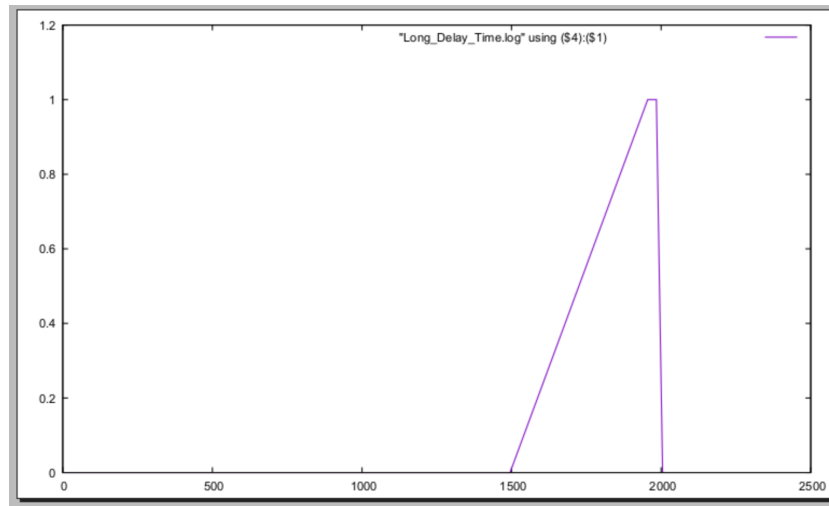


Рис. 3.17: Периоды времени, когда значения задержки в очереди превышали заданное значение

## 4 Выводы

Мы реализовали модель  $M|M|1|\infty$  в *CPN tools*.

## Список литературы

1. Королькова А.В., Кулябов Д.С. Лабораторная работа 11. Модель системы массового обслуживания М/М/1/ [Электронный ресурс].
2. Королькова А.В., Кулябов Д.С. Сети Петри. Моделирование в CPN Tools [Электронный ресурс].