

Лабораторная работа №11

Модель системы массового обслуживания $M|M|1$

Астраханцева А. А.

Содержание

1	Введение	4
1.1	Цель работы	4
1.2	Задание	4
2	Выполнение лабораторной работы	5
2.1	Описание модели	5
2.2	Мониторинг параметров моделируемой системы	11
3	Выводы	18
	Список литературы	19

Список иллюстраций

2.1	Граф сети системы обработки заявок в очереди	6
2.2	Граф генератора заявок системы	6
2.3	Граф процесса обработки заявок на сервере системы	6
2.4	Задание деклараций системы	8
2.5	Параметры элементов основного графа системы обработки заявок в очереди	8
2.6	Параметры элементов генератора заявок системы	9
2.7	Параметры элементов обработчика заявок системы	11
2.8	Функция Predicate монитора Ostanovka	11
2.9	Функция Observer монитора Queue Delay	12
2.10	Файл Queue_Delay.log	12
2.11	График изменения задержки в очереди	13
2.12	Функция Observer монитора Queue Delay Real	14
2.13	Содержимое Queue_Delay_Real.log	14
2.14	Функция Observer монитора Long Delay Time	15
2.15	Определение longdelaytime в декларациях	15
2.16	Содержимое Long_Delay_Time.log	16
2.17	Периоды времени, когда значения задержки в очереди превышали заданное значение	17

1 Введение

1.1 Цель работы

Реализовать модель $M|M|1$ в CPN tools.

1.2 Задание

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

2 Выполнение лабораторной работы

2.1 Описание модели

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 систем (рис. 2.1), на втором — генератор заявок (рис. 2.2), на третьем — сервер обработки заявок (рис. 2.3).

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.

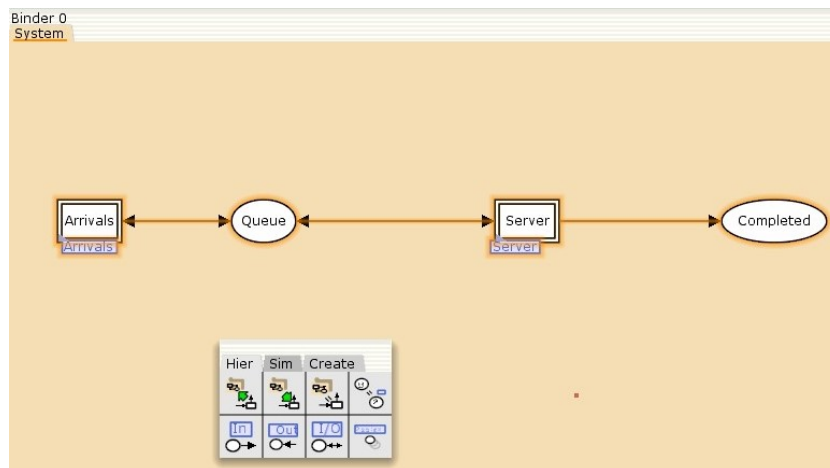


Рис. 2.1: Граф сети системы обработки заявок в очереди

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).

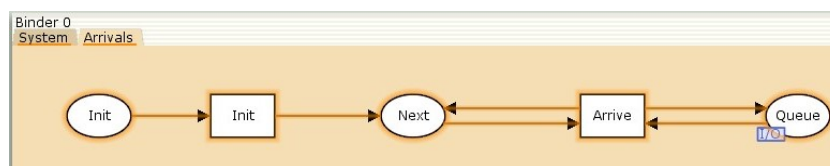


Рис. 2.2: Граф генератора заявок системы

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).

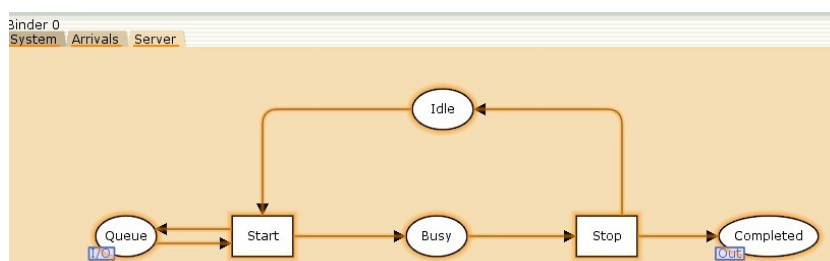


Рис. 2.3: Граф процесса обработки заявок на сервере системы

Зададим декларации системы (рис. 2.4).

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).

```

▼ 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 * J
    ▼ var proctime : INT;
    ▼ var job : Job;
    ▼ var jobs : Jobs;
    ► fun expTime
    ▼ fun intTime() = IntInf.toInt (time());
    ▼ fun newJob() = {jobType = JobType.ra
                      AT      = intTime()}

  ▼ Standard declarations
    ► colset BOOL
    ► colset STRING

```

Рис. 2.4: Задание деклараций системы

Зададим параметры модели на графах сети.

На листе System (рис. 2.5):

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



Рис. 2.5: Параметры элементов основного графа системы обработки заявок в очереди

На листе Arrivals (рис. 2.6):

- 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]` задаёт поступление заявки в очередь;
- на дуге от позиции `Queue` к переходу `Arrive` выражение `jobs` задаёт обратную связь.

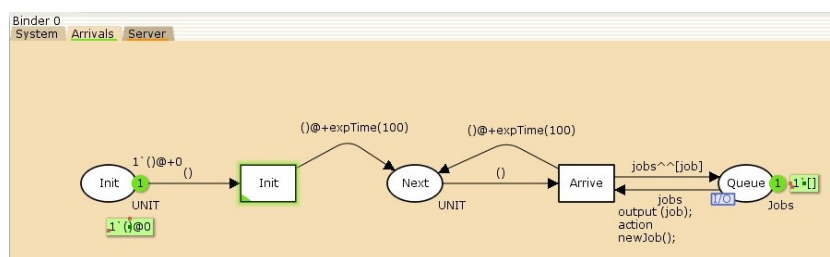


Рис. 2.6: Параметры элементов генератора заявок системы

На листе Server (рис. 2.7):

- у позиции `Busy`: множество цветов фишек — `Server`, начальное значение маркировки — `1`server@0` 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.

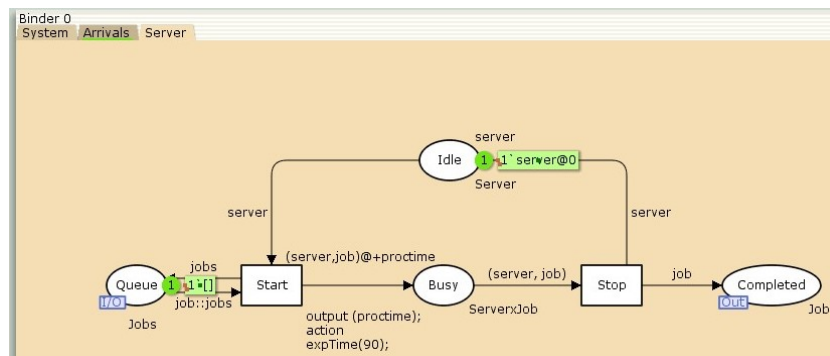


Рис. 2.7: Параметры элементов обработчика заявок системы

2.2 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. 2.8):

```

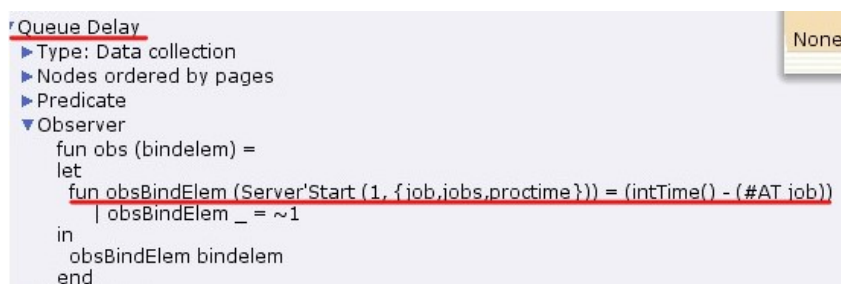
▼ Ostanovka
  Type: Break point
  ▶ Nodes ordered by pages
  ▼ Predicate
    fun pred (bindelem) =
    let
      fun predBindElem (Server'Start (1,
        {job,jobs,proctime})) = Queue_Delay.count()=200
        | predBindElem _ = false
    in
      predBindElem bindelem
    end
  
```

Рис. 2.8: 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. 2.9):



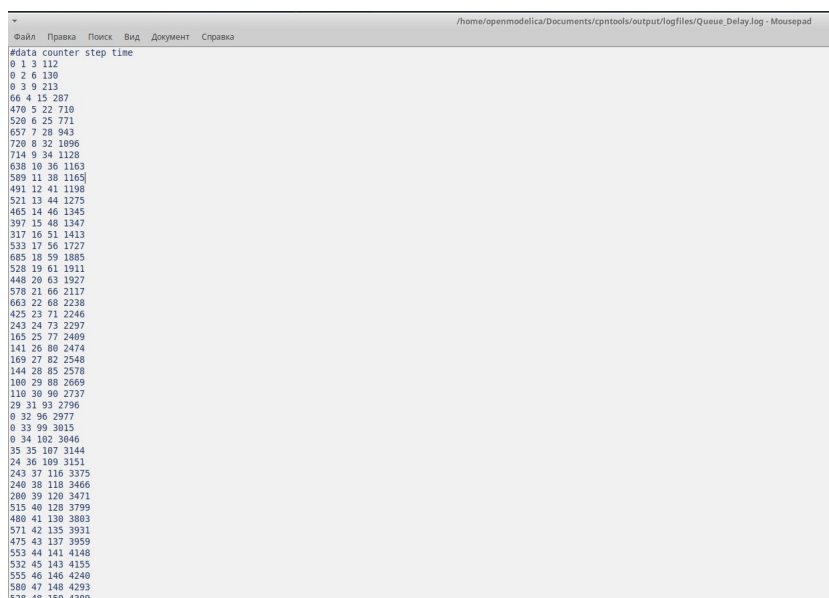
```

Queue Delay
Type: Data collection
Nodes ordered by pages
Predicate
Observer
  fun obs (bindelem) =
  let
    fun obsBindElem (Server'Start (1, {job,jobs,proctime})) = (intTime() - (#AT job))
    | obsBindElem _ = ~1
  in
    obsBindElem bindelem
  end

```

Рис. 2.9: Функция Observer монитора Queue Delay

После запуска программы на выполнение в каталоге с кодом программы появится файл `Queue_Delay.log` (рис. 2.10), содержащий в первой колонке — значение задержки очереди, во второй — счётчик, в третьей — шаг, в четвёртой — время.



```

#data counter step time
0 1 3 112
0 2 6 130
0 3 9 213
66 4 15 287
476 5 22 710
526 6 25 771
657 7 28 943
720 8 32 1096
714 9 34 1128
638 10 36 1163
589 11 38 1185
491 12 41 1198
521 13 44 1275
465 14 46 1345
397 15 48 1347
317 16 51 1413
533 17 56 1727
685 18 59 1885
528 19 61 1911
448 20 63 1927
578 21 66 2117
663 22 68 2238
425 23 71 2246
243 24 73 2297
165 25 77 2409
141 26 80 2474
169 27 82 2548
144 28 85 2578
180 29 88 2669
110 30 90 2737
29 31 93 2796
0 32 96 2977
0 33 99 3015
0 34 102 3046
35 35 107 3144
24 36 109 3151
243 37 116 3375
240 38 118 3466
280 39 120 3471
515 40 128 3799
480 41 130 3803
571 42 135 3931
475 43 137 3959
553 44 141 4148
532 45 143 4155
555 46 146 4240
580 47 148 4293
528 48 150 4389

```

Рис. 2.10: Файл Queue_Delay.log

С помощью `gnuplot` можно построить график значений задержки в очереди (рис. 2.11), выбрав по оси *x* время, а по оси *y* — значения задержки:

```
gnuplot
set terminal png
set output "plot.png"
plot "Queue_Delay.log" using ($4):($1) with lines
quit
```

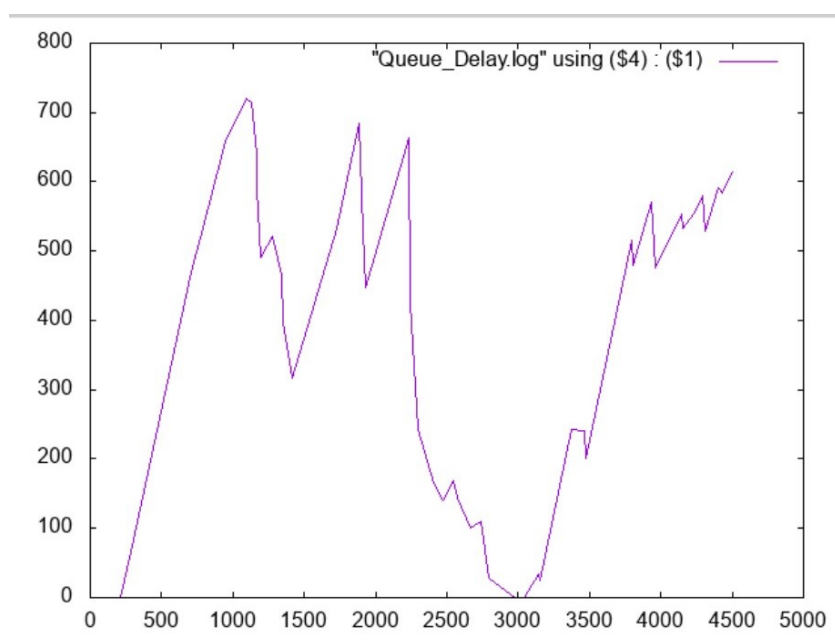


Рис. 2.11: График изменения задержки в очереди

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. 2.12):

```

Queue Delay Real
  Type: Data collection
  Nodes ordered by pages
  Predicate
  Observer
    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. 2.12: 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. 2.13):

```

#data counter step time
0.000000 1 3 174
0.000000 2 6 285
0.000000 3 9 480
452.000000 4 19 1012
412.000000 5 21 1041
366.000000 6 23 1075
369.000000 7 25 1109
267.000000 8 27 1111
298.000000 9 29 1175
173.000000 10 31 1182
311.000000 11 35 1321
246.000000 12 40 1459
214.000000 13 42 1513
204.000000 14 44 1545
175.000000 15 46 1587
175.000000 16 48 1588
102.000000 17 52 1793
99.000000 18 56 1816
210.000000 19 59 2004
337.000000 20 64 2153
282.000000 21 66 2221
213.000000 22 68 2237
171.000000 23 70 2244
286.000000 24 74 2371
66.000000 25 76 2377
145.000000 26 80 2485
205.000000 27 82 2602
241.000000 28 84 2648
85.000000 29 88 2831
26.000000 30 90 2835
0.000000 31 93 2994
444.000000 32 101 3448
399.000000 33 103 3472
102.000000 34 105 3489

```

Рис. 2.13: 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. 2.14):

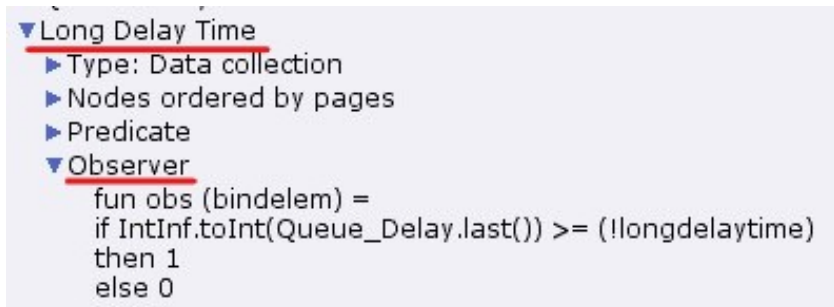


Рис. 2.14: Функция Observer монитора Long Delay Time

При этом необходимо в декларациях задать глобальную переменную (в форме ссылки на число 200): longdelaytime (рис. 2.15).



Рис. 2.15: Определение longdelaytime в декларациях

После запуска программы на выполнение в каталоге с кодом программы появится файл Long_Delay_Time.log (рис. 2.16)

```
▼ /home/openmodelica/Documents/cpntools/output/logfiles/Long_Delay_Time.log - Mousep: -  
Файл  Правка  Поиск  Вид  Документ  Справка  
#data counter step time  
0 1 3 114  
0 2 6 216  
0 3 9 433  
0 4 12 718  
0 5 15 1077  
0 6 18 1160  
0 7 21 1306  
0 8 24 1440  
0 9 28 1616  
0 10 30 1619  
0 11 33 1659  
0 12 36 1702  
0 13 41 1916  
0 14 43 1987  
0 15 46 2068  
0 16 48 2112  
0 17 53 2267  
0 18 56 2328  
0 19 60 2382  
0 20 62 2383  
0 21 64 2453  
1 22 69 2587  
0 23 72 2630  
0 24 74 2710  
1 25 79 2892  
1 26 83 3127  
1 27 85 3159  
1 28 89 3394  
1 29 91 3401  
1 30 93 3423  
1 31 100 3608  
1 32 104 3817  
1 33 106 3955  
1 34 108 3966  
1 35 110 3998  
1 36 113 4021  
1 37 115 4106  
1 38 117 4183  
1 39 119 4378  
1 40 121 4388  
1 41 123 4418  
0 42 128 4560  
0 43 130 4568  
0 44 133 4596  
0 45 137 4768  
1 46 141 4891  
1 47 143 5006
```

Рис. 2.16: Содержимое Long_Delay_Time.log

С помощью gnuplot можно построить график (рис. 2.17), демонстрирующий, в какие периоды времени значения задержки в очереди превышали заданное значение 200.

```
gnuplot  
set termnal png  
set output "plot2.png"
```



```
set yrange [0:1.2]
plot [0:] [0:1.2] "Long_Delay_Time.log" using ($4):($1) with lines
quit
```

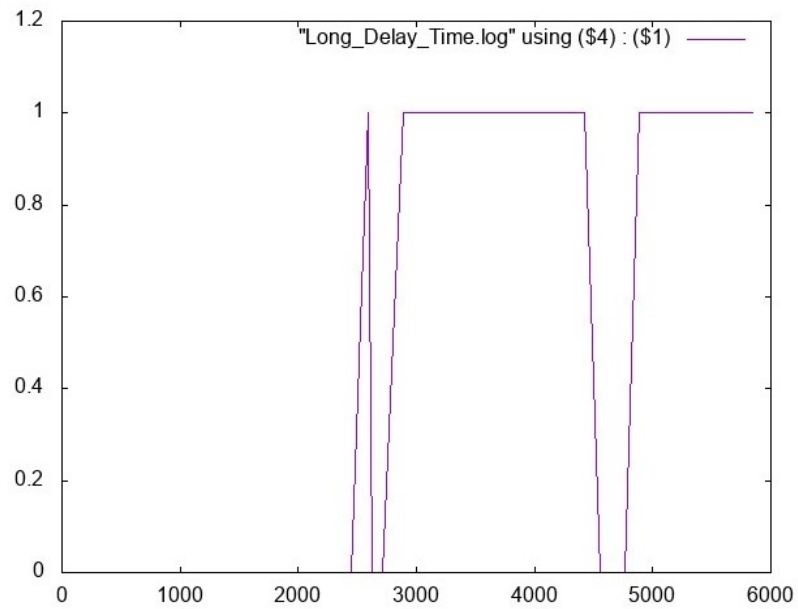


Рис. 2.17: Периоды времени, когда значения задержки в очереди превышали заданное значение

3 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. Моделирование информационных процессов. Модель системы массового обслуживания M|M|1 - 2025. — 10 с.
2. Modeling with Coloured Petri Nets [Электронный ресурс] // URL: <https://cpntools.org/2018/01/started>.
3. Jensen K., Kristensen L.M., Wells L. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems // Software Tools for Technology Transfer. 2007. — URL: https://cs.au.dk/fileadmin/site_files/cs/research_areas/centers_and_projects/CPNTools/CPNTools.pdf.
4. Ratzer A.V., Wells L., Lassen H.M., et al. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets // ICATPN Proceedings, 2003 — URL: <https://api.semanticscholar.org/CorpusID:12059006>.
5. Beaudouin-Lafon M., Mackay W.E., Andersen P., et al. Editing and Simulating Coloured Petri Nets // CPNTools.doc, University of Aarhus, 2000 — URL: <https://www.lri.fr/~mbl/papers/PN2000/paper.pdf>.