

Лабораторная работа 11

Модель системы массового обслуживания $M|M|1$

Извекова Мария Петровна

Содержание

Цель работы	5
Задание	6
Выполнение лабораторной работы	7
Мониторинг параметров моделируемой системы	13
Выводы	22

Список иллюстраций

Список таблиц

Цель работы

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

Задание

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

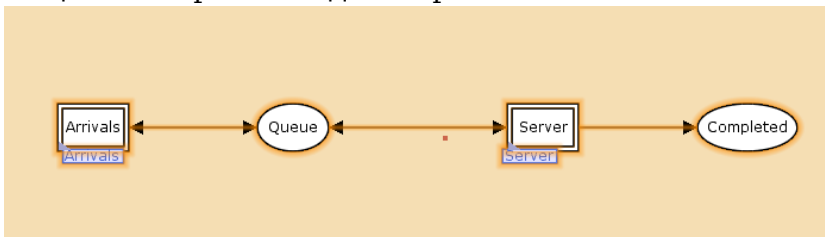
Выполнение лабораторной работы

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. [-@fig:001]), на втором — генератор заявок (рис. [-@fig:002]), на третьем — сервер обработки заявок (рис. [-@fig:003]).

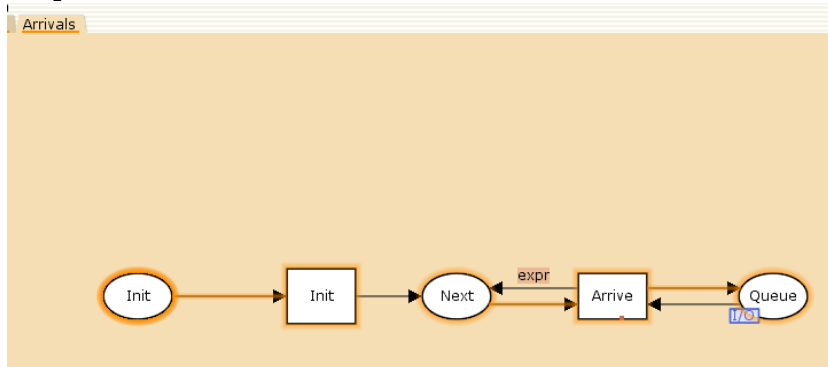
Сеть имеет 2 позиции (очередь — Queue, обслуженные заявки — Complited) и два перехода (генерировать заявку — Arrivals, передать заявку на обработку серверу — Server). Переходы имеют сложную иерархическую структуру, задаваемую на отдельных листах модели (с помощью соответствующего инструмента меню — Hierarchy).

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.

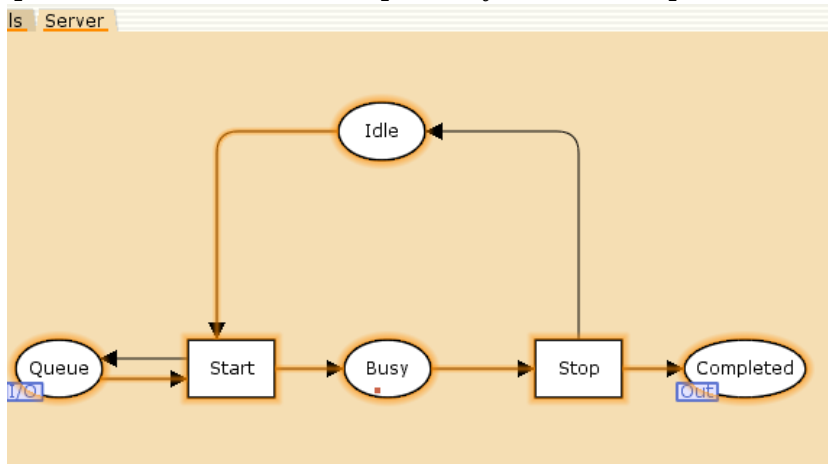


Граф генератора заявок имеет 3 позиции (текущая заявка — Init, следующая заявка — Next, очередь — Queue из листа System) и 2 перехода (Init — определяет распределение поступления заявок по экспоненциальному закону с интенсив-

ностью 100 заявок в единицу времени, Arrive — определяет поступление заявок в очередь).



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



Зададим декларации системы (рис. [-@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 — 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).

```

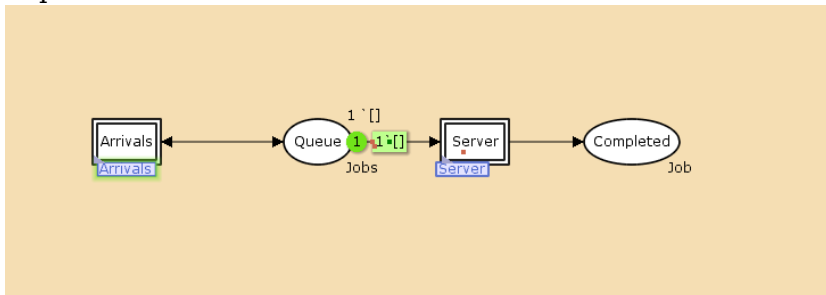
▼ queue_system.cpn
  Step: 0
  Time: 0
  ▶ Options
  ▶ History
  ▼ Declarations
    ▶ Standard declarations
    ▼ System
      ▼ colset MYUNIT = unit timed;
      ▼ colset INT = int;
      ▼ colset Server = with server timed;
      ▼ colset JobType = with A|B;
      ▼ colset Job = record
        jobType: 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()}
    ▶ Monitors
    ▼ System
      Arrivals
      Server

```

Зададим параметры модели на графах сети.

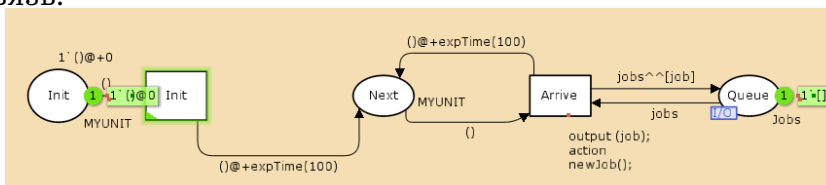
На листе System (рис. [-@fig:005]):

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. [-@fig:006]):

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 jobs¹ задаёт поступление заявки в очередь; на дуге от позиции Queue к переходу Arrive выражение jobs задаёт обратную связь.

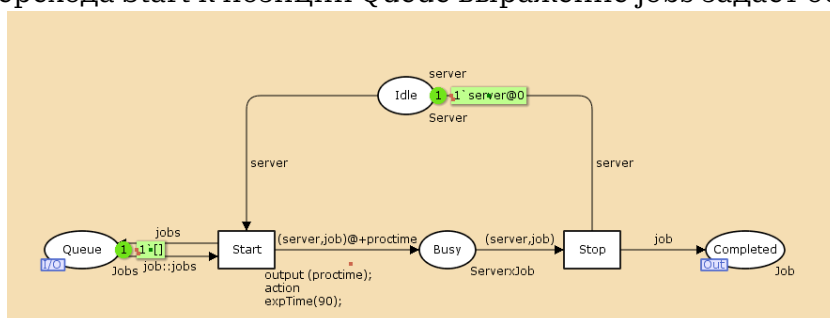


На листе Server (рис. [-@fig:007]):

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

¹job

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.

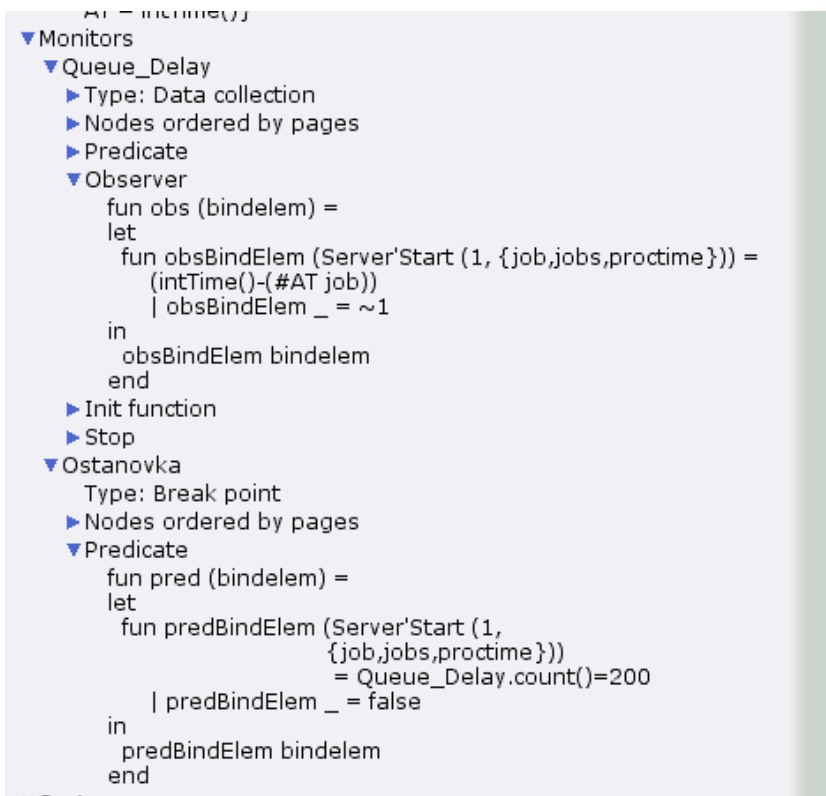


Мониторинг параметров моделируемой системы

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`.

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), подчеркивание 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. [-@fig:008]):



После запуска программы на выполнение в каталоге с кодом программы появится файл Queue_Delay.log (рис. [-@fig:009]), содержащий в первой колонке — значение задержки очереди, во второй — счётчик, в третьей — шаг, в четвёртой — время.

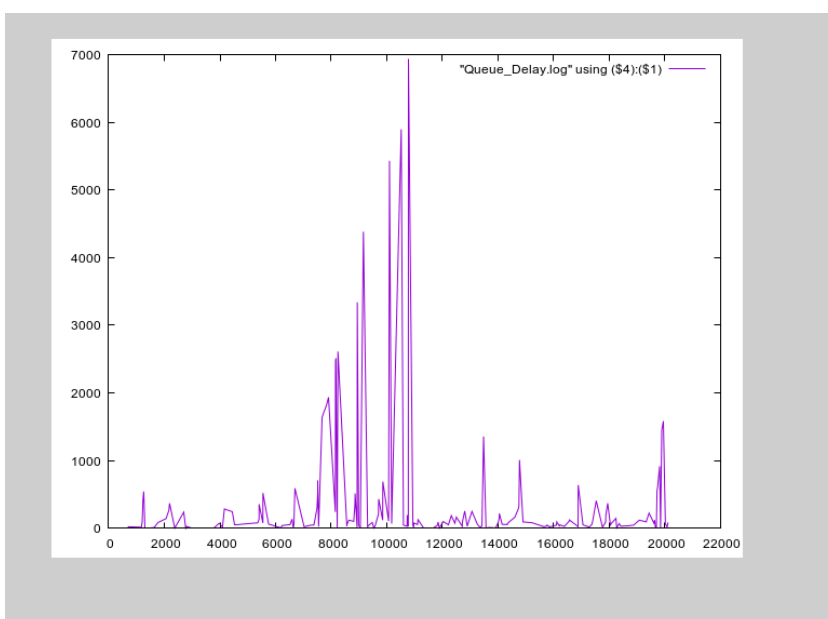
step	time	counter	data	time
1	#data	counter	step	time
2	0	1	3	43
3	100	2	6	231
4	0	3	9	447
5	85	4	13	651
6	33	5	18	823
7	2	6	21	893
8	56	7	24	1081
9	315	8	26	1090
10	374	9	28	1093
11	790	10	30	1287
12	0	11	33	1319
13	257	12	39	1682
14	45	13	42	1751
15	364	14	44	1779
16	32	15	50	1869
17	43	16	53	1921
18	118	17	55	1935
19	193	18	57	2000
20	71	19	63	2334
21	258	20	65	2417
22	361	21	67	2489
23	408	22	69	2489
24	7	23	73	2535
25	17	24	75	2542
26	787	25	77	2581
27	29	26	80	2634
28	1259	27	82	2642
29	1520	28	84	2861
30	0	29	87	2918
31	0	30	90	3004
32	30	31	94	3116
33	173	32	96	3231
34	35	33	100	3307
35	6	34	104	3394
36	96	35	106	3408
37	155	36	108	3424
38	45	37	112	3601
39	200	38	114	3627
40	19	39	118	3659
41	77	40	120	3707
42	0	41	123	3789
43	14	42	126	3881
44	0	43	129	3910
45	0	44	132	4043
46	38	45	136	4099
47	99	46	138	4142
48	53	47	141	4248
49	12	48	144	4365
50	55	49	149	4669
51	125	50	151	4711
52	209	51	153	4740
53	10	52	157	4874
54	11	53	160	4893

С помощью gnuplot можно построить график значений задержки в очереди (рис. [-@fig:010]), выбрав по оси x время, а по оси y — значения задержки:

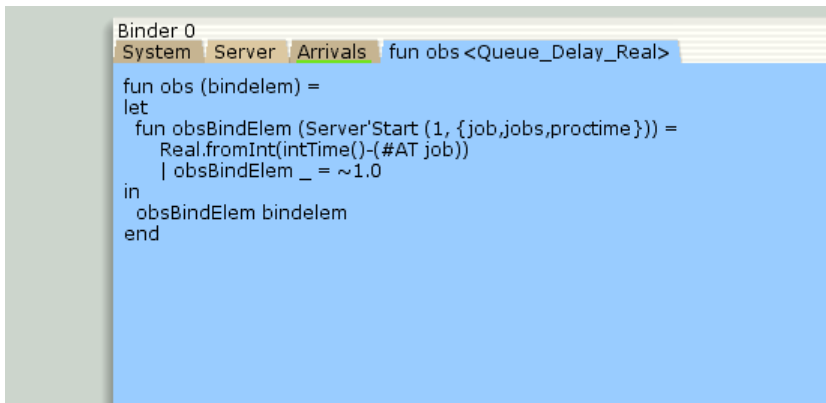
```
#!/usr/bin/gnuplot -persist

set encoding utf8
set term pngcairo font "Helvetica,9"

set out 'win_1.png'
plot "Queue_Delay.log" using ($4):($1) with lines
```



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. [-@fig:011]):

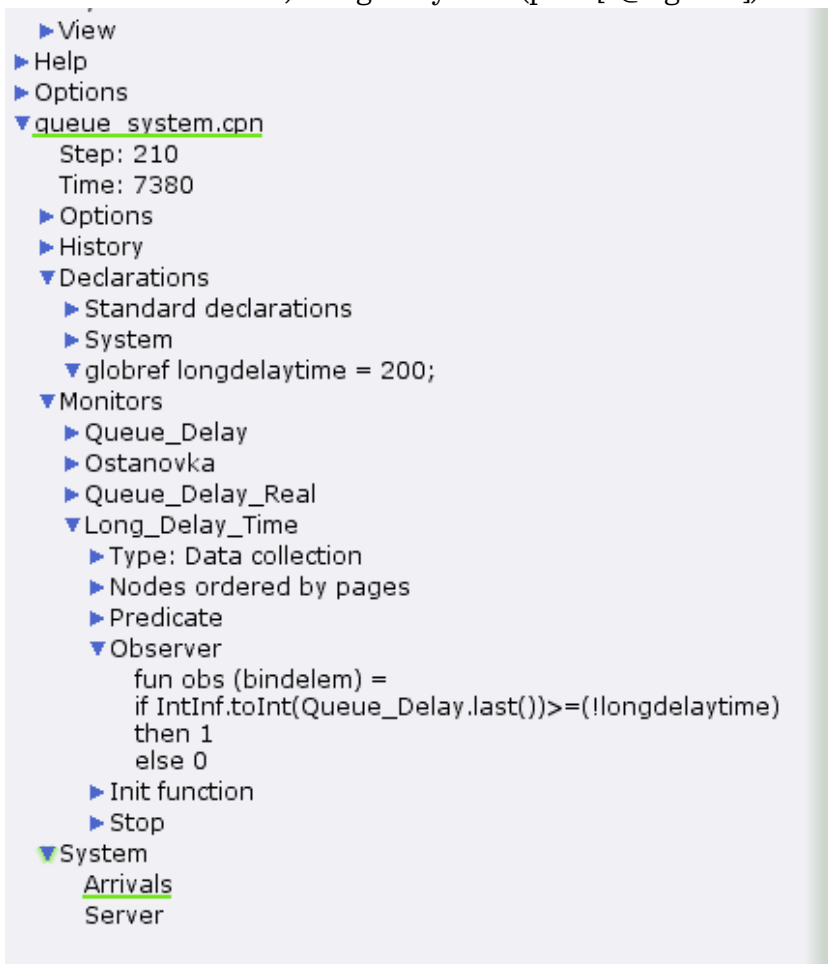
A screenshot of a NetLogo code editor window. The title bar says "Binder 0". Below the title bar, there are four tabs: "System", "Server", "Arrivals", and "fun obs <Queue_Delay_Real>". The "Arrivals" tab is currently selected. The code in the editor is as follows:

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

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. [-@fig:012]):

Файл	Правка	Поиск	Вид	Док
1	#data	counter	step	time
2	0.000000	1	3	43
3	100.000000	2	6	231
4	0.000000	3	9	447
5	85.000000	4	13	651
6	33.000000	5	18	823
7	2.000000	6	21	893
8	56.000000	7	24	1081
9	315.000000	8	26	1090
10	374.000000	9	28	1093
11	790.000000	10	30	1287
12	0.000000	11	33	1319
13	257.000000	12	39	1682
14	45.000000	13	42	1751
15	364.000000	14	44	1779
16	32.000000	15	50	1869
17	43.000000	16	53	1921
18	118.000000	17	55	1935
19	193.000000	18	57	2000
20	71.000000	19	63	2334
21	258.000000	20	65	2417
22	361.000000	21	67	2489
23	408.000000	22	69	2489
24	7.000000	23	73	2535
25	17.000000	24	75	2542
26	787.000000	25	77	2581
27	29.000000	26	80	2634
28	1259.000000	27	82	2642
29	1520.000000	28	84	2861
30	0.000000	29	87	2918
31	0.000000	30	90	3004
32	30.000000	31	94	3116
33	173.000000	32	96	3231
34	35.000000	33	100	3307
35	6.000000	34	104	3394
36	96.000000	35	106	3408
37	155.000000	36	108	3424
38	45.000000	37	112	3601
39	200.000000	38	114	3627
40	19.000000	39	118	3659
41	77.000000	40	120	3707
42	0.000000	41	123	3789
43	14.000000	42	126	3881
44	0.000000	43	129	3910
45	0.000000	44	132	4043
46	38.000000	45	136	4099
47	99.000000	46	138	4142
48	53.000000	47	141	4248
49	12.000000	48	144	4365
50	55.000000	49	149	4669
51	125.000000	50	151	4711
52	209.000000	51	153	4740
53	10.000000	52	157	4874
54	11.000000	53	160	4893

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. При этом необходимо в декларациях задать глобальную переменную (в форме ссылки на число 200): longdelaytime (рис. [-@fig:013]).

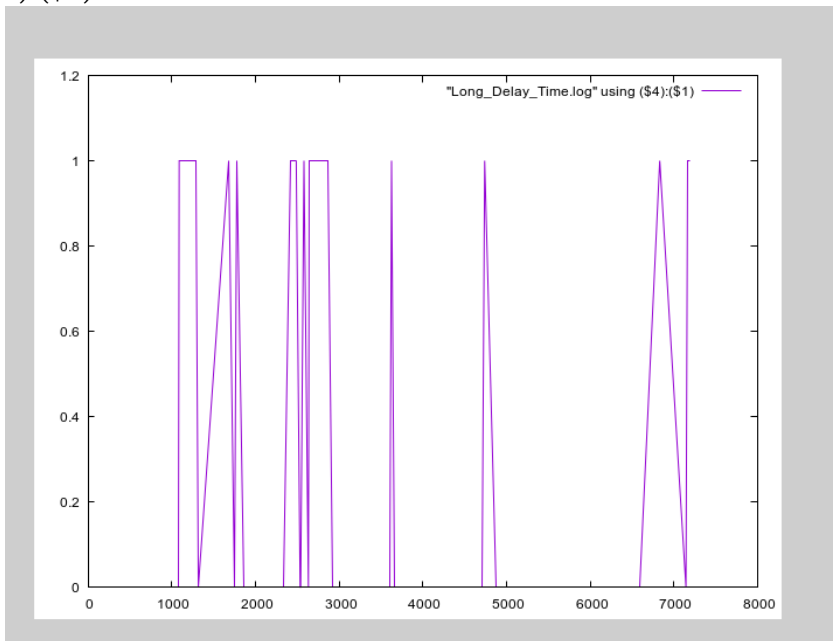


После запуска программы на выполнение в каталоге с кодом программы появится файл Long_Delay_Time.log (рис. [-@fig:014])

Файл	Правка	Поиск
1	#data counter ste	
2	0 1 3 43	
3	0 2 6 231	
4	0 3 9 447	
5	0 4 13 651	
6	0 5 18 823	
7	0 6 21 893	
8	0 7 24 1081	
9	1 8 26 1090	
10	1 9 28 1093	
11	1 10 30 1287	
12	0 11 33 1319	
13	1 12 39 1682	
14	0 13 42 1751	
15	1 14 44 1779	
16	0 15 50 1869	
17	0 16 53 1921	
18	0 17 55 1935	
19	0 18 57 2000	
20	0 19 63 2334	
21	1 20 65 2417	
22	1 21 67 2489	
23	1 22 69 2489	
24	0 23 73 2535	
25	0 24 75 2542	
26	1 25 77 2581	
27	0 26 80 2634	
28	1 27 82 2642	
29	1 28 84 2861	
30	0 29 87 2918	
31	0 30 90 3004	
32	0 31 94 3116	
33	0 32 96 3231	
34	0 33 100 3307	
35	0 34 104 3394	
36	0 35 106 3408	
37	0 36 108 3424	
38	0 37 112 3601	
39	1 38 114 3627	
40	0 39 118 3659	
41	0 40 120 3707	
42	0 41 123 3789	
43	0 42 126 3881	
44	0 43 129 3910	
45	0 44 132 4043	
46	0 45 136 4099	
47	0 46 138 4142	
48	0 47 141 4248	
49	0 48 144 4365	
50	0 49 149 4669	
51	0 50 151 4711	
52	1 51 153 4740	
53	0 52 157 4874	
--	--	----

С помощью `gnuplot` можно построить график (рис. [-@fig:015]), демонстрирующий, в какие периоды времени значения задержки в очереди превышали заданное значение 200.

```
#!/usr/bin/gnuplot -persist set encoding utf8 set term pngcairo font "Helvetica,9"
set out 'win_3.png' set style line 2 plot [0:] [0:1.2] "Long_Delay_Time.log" using
($4):($1) with lines
```



Выводы

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