

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

## ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №11

дисциплина: Моделирование информационных процессов

Студент:

Ibragimov Ulugbek

Группа:

НФИбд-02-20

МОСКВА

2023 г.

Цель .....	3
Задачи .....	4
Выполнение лабораторной работы.....	5
Анализ результатов .....	12
Вывод.....	13

## **Цель**

Приобретение и улучшение навыков моделирования при помощи такого средства, как CPN Tools. Обрести необходимые теоретические знания о сетях Петри и практические навыки их построения.

## Задачи

1. Построить модель системы массового обслуживания: 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.
2. Провести мониторинг параметров моделируемой системы

## Выполнение лабораторной работы

1. Создадим новую сеть и зададим для нее декларации: типы, переменные, начальные значения, функции. (Рис. 1)



Рис.1. Декларации модели системы массового обслуживания

The diagram shows a Petri net model of a queue system. It consists of four places: Arrivals, Queue, Server, and Completed. The Arrivals place is connected to the Queue place by a transition with a rate of 1 and an empty guard []. The Queue place is connected to the Server place by a transition with a rate of 1 and a guard of jobs. The Server place is connected to the Completed place by a transition with a rate of 1 and a guard of job. The Queue place currently contains 1 token, as indicated by the green circle with the number 1.

The screenshot shows the CPN Tools interface with a Petri net model. The top bar indicates 'Binder 0'. Below it, there are tabs for 'System', 'Arrivals', and 'Server'. The main workspace displays a Petri net with the following components:

- Places:**
  - Init:** An oval place containing two tokens (green circles with numbers 1 and 1). It has an outgoing transition labeled '()' with a guard '1'()'@+0 and a unit 'UNIT'.
  - Next:** An oval place that receives tokens from the 'Init' place and the 'Arrive' transition. It has an outgoing transition to the 'Arrive' place labeled '()' with a guard '()'@+expTime(100) and a unit 'UNIT'.
  - Arrive:** A rectangular place that receives tokens from the 'Next' place and the 'Queue' place. It has an outgoing transition to the 'Queue' place labeled 'jobs' with a guard 'jobs'^^[job] and an action 'output (job); action newJob();'.
  - Queue:** An oval place containing one token (green circle with number 1). It has an outgoing transition to the 'Arrive' place labeled 'jobs' with a guard '1/0' and a unit 'Jobs'.
- Transitions:**
  - Init to Next:** Labeled '()' with guard '1'()'@+0 and unit 'UNIT'.
  - Next to Arrive:** Labeled '()' with guard '()'@+expTime(100) and unit 'UNIT'.
  - Arrive to Queue:** Labeled 'jobs' with guard 'jobs'^^[job] and action 'output (job); action newJob();'.
  - Queue to Arrive:** Labeled 'jobs' with guard '1/0' and unit 'Jobs'.

The diagram illustrates a queueing system in CPN Tools. It features the following components:

- Places:**
  - Queue:** Contains 1 token (green circle with '1').
  - Idle:** Contains 1 token (green circle with '1').
- Transitions:**
  - Start:** Triggered by the event 'jobs' from the Queue place. Guard:  $1'[]$ . Action:  $(server, job)@+proctime$ . Output: 'job::jobs' to the Queue place.
  - Busy:** Triggered by the event 'server' from the Idle place. Guard:  $(server, job)@+proctime$ . Action:  $output(proctime); action expTime(90);$ . Output: 'Server:Job' to the Stop transition.
  - Stop:** Triggered by the event 'job' from the Busy place. Output: 'job' to the Completed place.
  - Completed:** A final state place.
- Labels:**
  - Queue:** Labeled 'I/O'.
  - Completed:** Labeled 'Out'.

6

3. Запустим симуляцию. Для этого воспользуемся Tool box—>Simulation. (рис. 5)

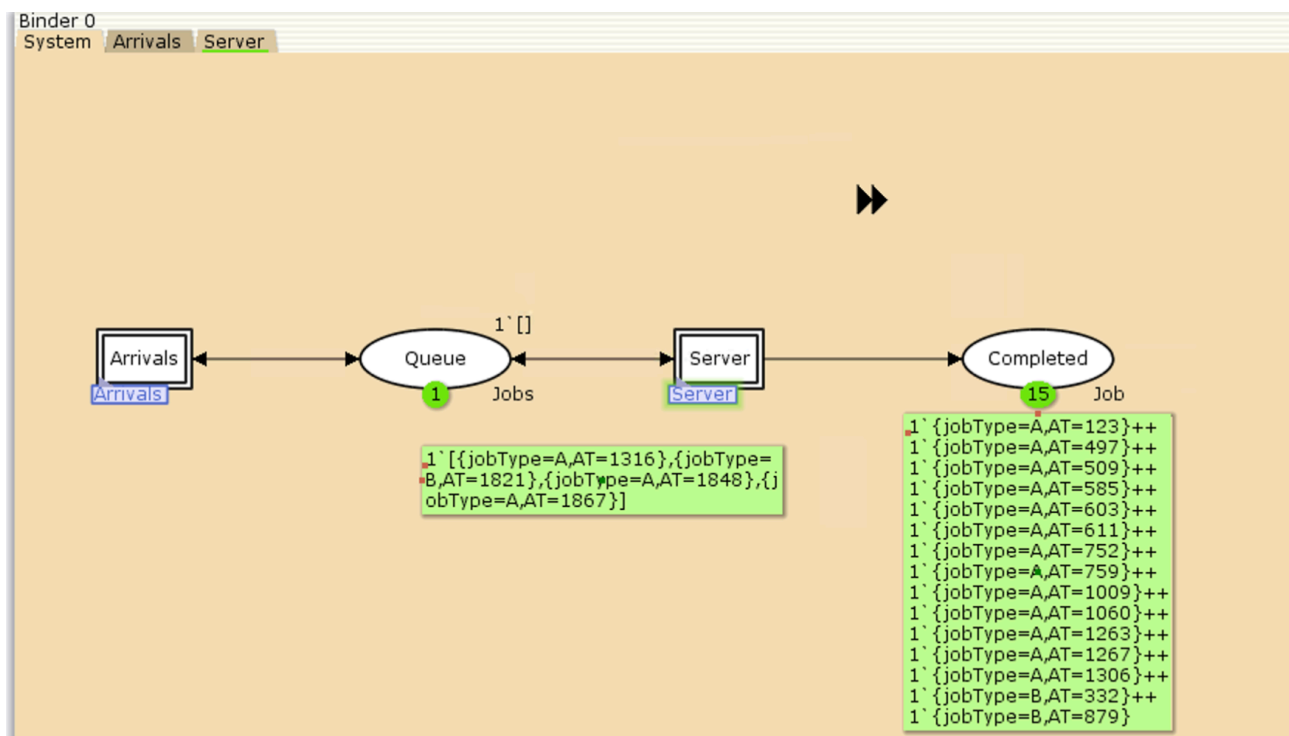


Рис.5. Симуляция модели «Обедающие мудрецы».

4. Произведем мониторинг параметров очереди. Поставим *Break Point* и устанавливаем её на переход *Start*. В разделе меню *Monitor* новый подраздел, назовём *Ostanovka*. В этом подразделе внесем изменения в функцию *Predicate*. (Рис. 6)

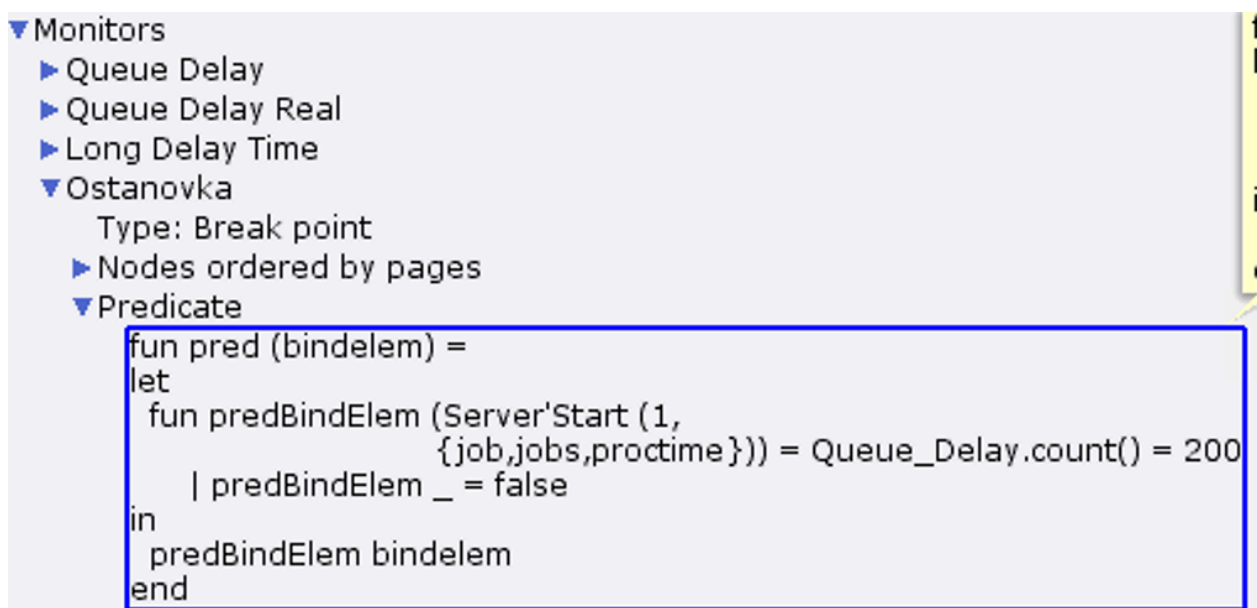


Рис. 6. Функция Predicate монитора Ostanovka

5. С помощью палитры *Monitoring* выбираем *Data Call* и устанавливаем на переходе *Start*. Появившийся в меню монитор называем *Queue Delay*. Изменяем функция *Observer* для *Queue Delay*. (Рис. 7)

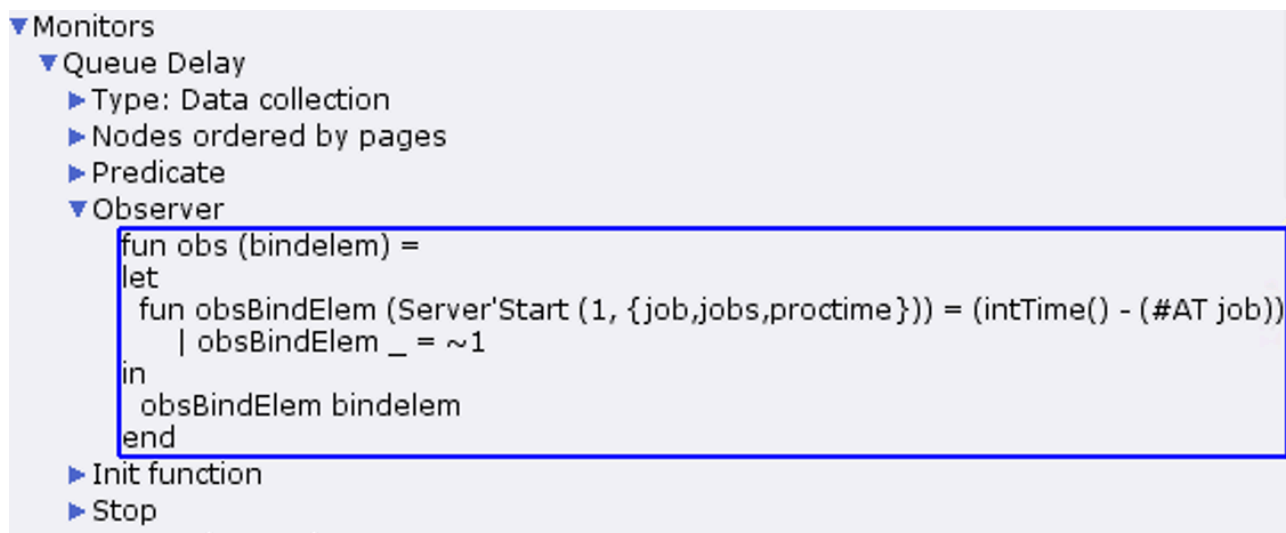


Рис. 7. Функция Observer монитора Queue Delay

6. Произведем симуляцию. В каталоге с кодом программы появился файл «*outputs/logfiles/Queue\_Delay.log*». Построим график, используя данный файл, при помощи GNUplot. (Рис. 8-9)

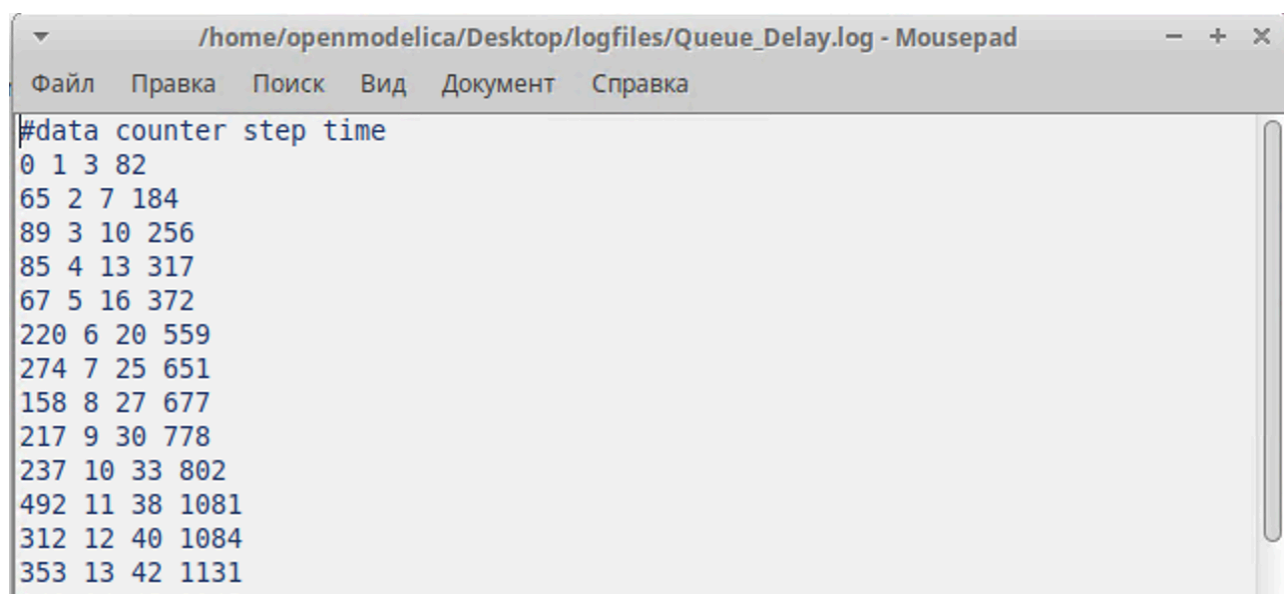


Рис.8. Фрагмент файла *Queue\_Delay.log*



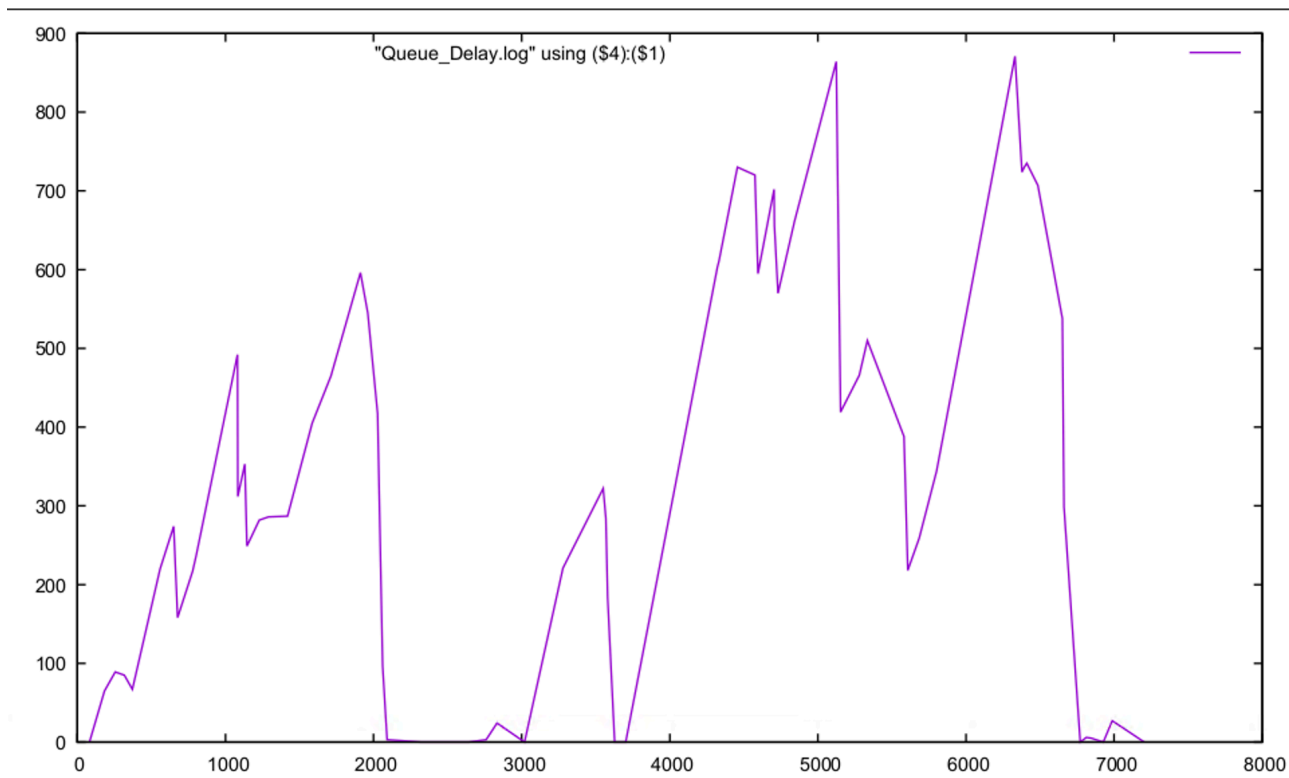


Рис. 9. График изменения задержки в очереди

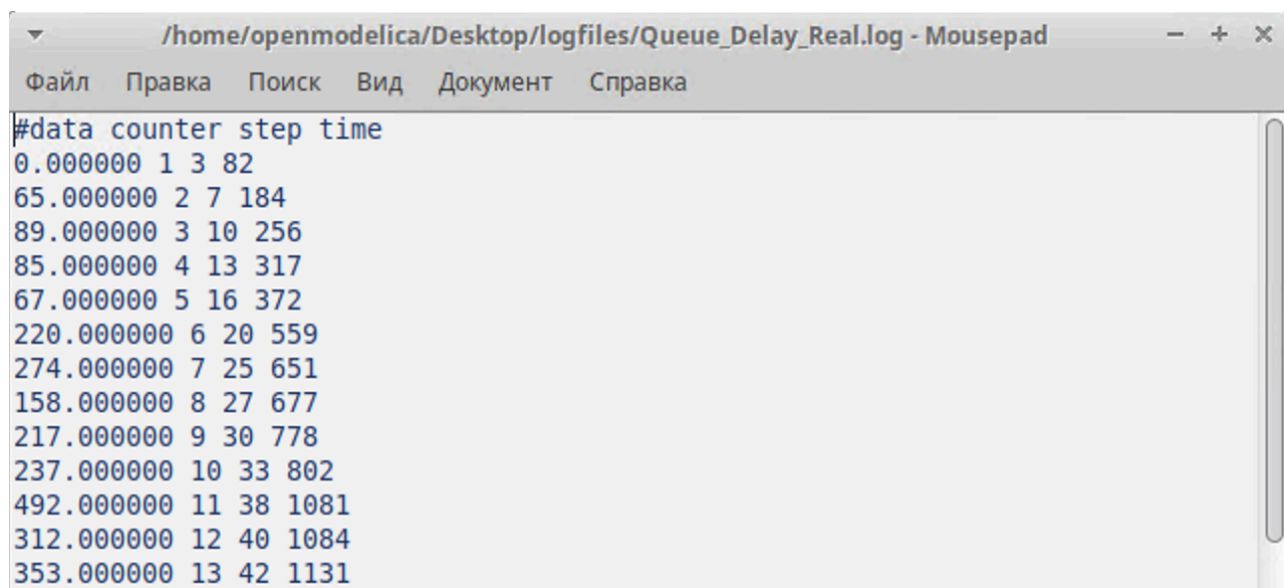
7. Посчитаем задержку в действительных значениях. Поставим еще один *Data Call* и устанавливаем на переходе *Start*. Появившийся в меню монитор называем *Queue Delay Real*. Изменяем функция *Observer* для данного монитора. (Рис. 10)

```

▼ Monitors
  ► Queue Delay
  ▼ 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
    ► Init function
    ► Stop
    ► Long Delay Time
    ► Ostanovka
  
```

Рис. 10. Функция Observer монитора Queue Delay Real

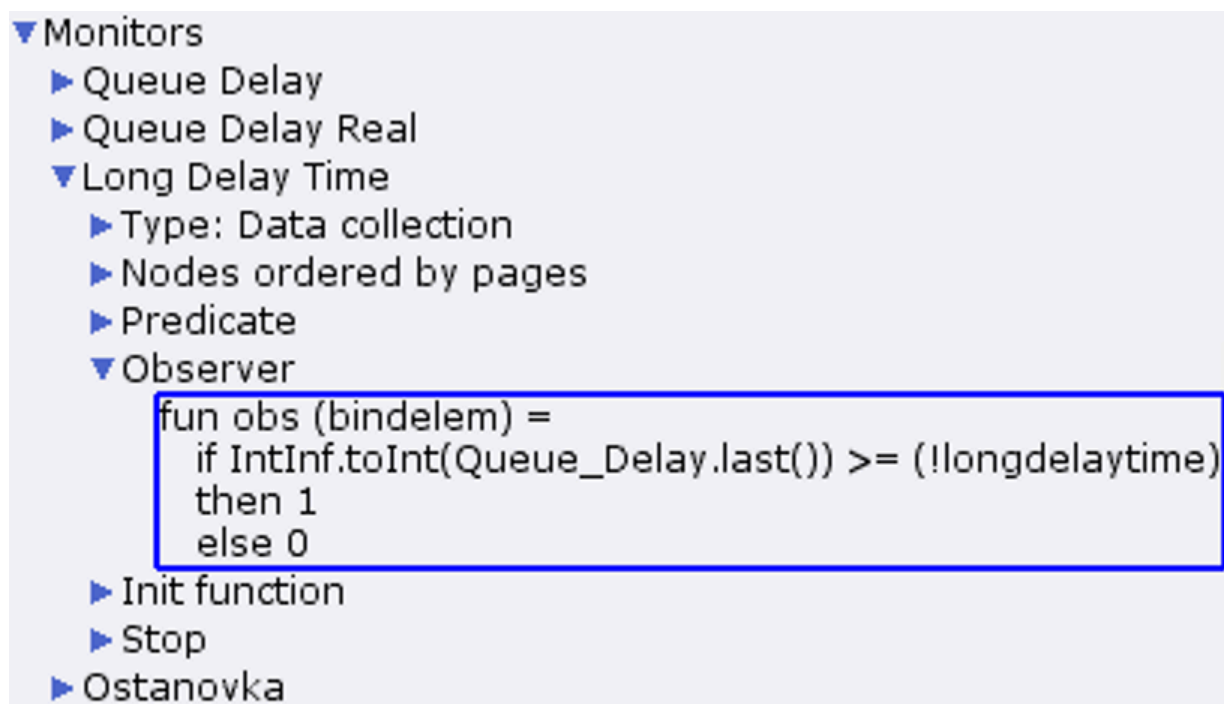
8. Произведем симуляцию. В каталоге с кодом программы появился файл «*outputs/logfiles/Queue\_Delay\_Real.log*». (Рис. 11)



```
#data counter step time
0.000000 1 3 82
65.000000 2 7 184
89.000000 3 10 256
85.000000 4 13 317
67.000000 5 16 372
220.000000 6 20 559
274.000000 7 25 651
158.000000 8 27 677
217.000000 9 30 778
237.000000 10 33 802
492.000000 11 38 1081
312.000000 12 40 1084
353.000000 13 42 1131
```

Рис.11. Фрагмент файла *Queue\_Delay\_Real.log*

9. Посчитаем, сколько раз задержка превысила заданное значение. С помощью палитры *Monitoring* выбираем *Data Call* и устанавливаем на переходе *Start*. Монитор называем *Long Delay Time*. Изменяем функция *Observer* для данного монитора. (Рис. 12)



```
▼ Monitors
  ► Queue Delay
  ► Queue Delay Real
  ▼ Long Delay Time
    ► Type: Data collection
    ► Nodes ordered by pages
    ► Predicate
    ▼ Observer
      fun obs (bindelem) =
        if IntInf.toInt(Queue_Delay.last()) >= (!longdelaytime)
        then 1
        else 0
    ► Init function
    ► Stop
  ► Ostanovka
```

Рис. 12. Функция Observer монитора *Long Delay Time*.

10. Произведем симуляцию. В каталоге с кодом программы появился файл «*outputs/logfiles/Long\_Delay\_Time.log*». Построим график, используя данный файл, при помощи GNUplot. (Рис. 13-14)

```

/home/openmodelica/Desktop/logfiles/Long_Delay_Time.log - Mousepad
Файл  Правка  Поиск  Вид  Документ  Справка
#data counter step time
0 1 3 82
0 2 7 184
0 3 10 256
0 4 13 317
0 5 16 372
1 6 20 559
1 7 25 651
0 8 27 677
1 9 30 778
1 10 33 802
1 11 38 1081
1 12 40 1084
1 13 42 1131

```

Рис.13. Фрагмент файла *Long\_Delay\_Time.log*

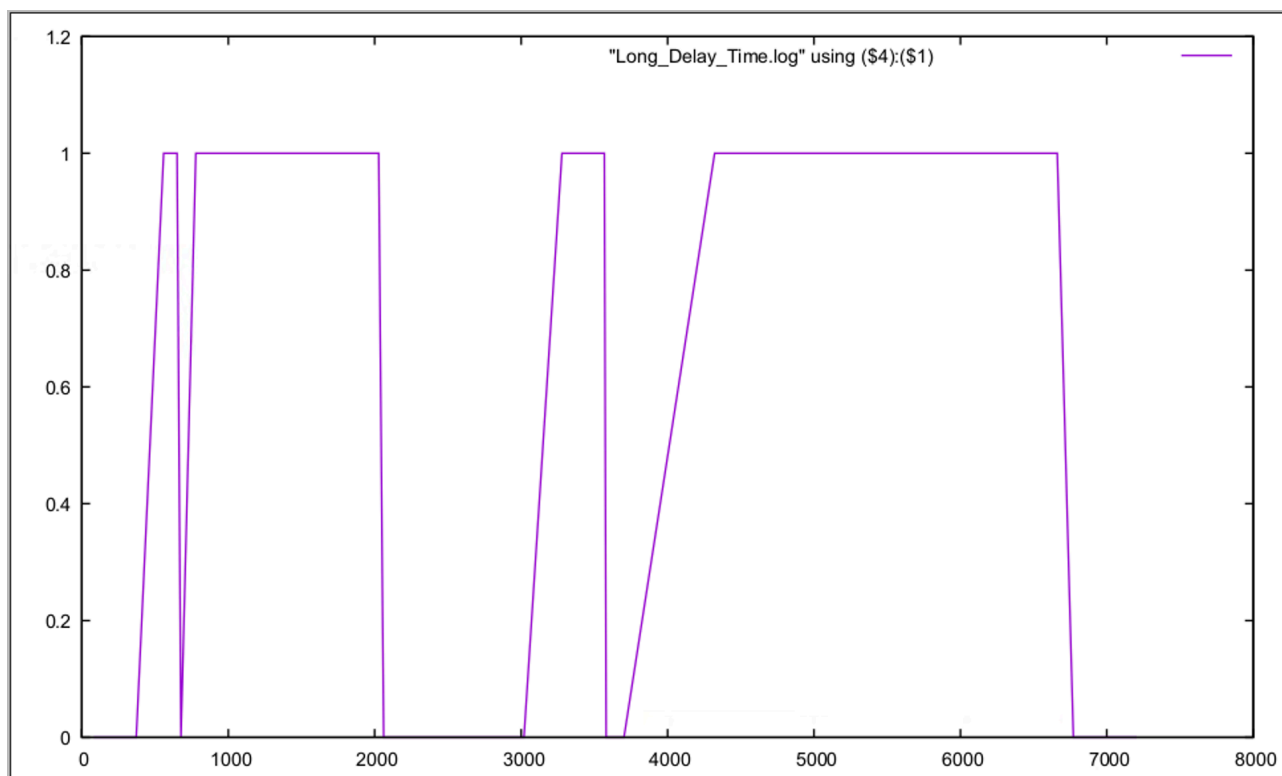


Рис. 14. Периоды времени, когда значения задержки в очереди превышали заданное значение.

## **Анализ результатов**

При выполнении работы не произошло непредвиденных проблем, работа была выполнена в соответствии с руководством. Стоит отметить, что моделирование при помощи CPN Tools происходит крайне комфортно, Tool box интуитивно понятен. Также само моделирование довольно быстро выполняется.

## **Вывод**

В результате выполнения работы, были приобретены и улучшены практические навыки построения сетей Петри при помощи специализированного средства CPN Tools. Была реализована модель системы массового обслуживания. Произведен мониторинг параметров системы.