

Лабораторная работа 11

Модель системы массового обслуживания $M|M|1$

Оразгелдиев Язгелди

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	20

Список иллюстраций

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

Список таблиц

1 Цель работы

Реализовать модель $M|M|1$ в CPNTools

2 Задание

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

3 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, на втором — генератор заявок, на третьем — сервер обработки заявок. 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.

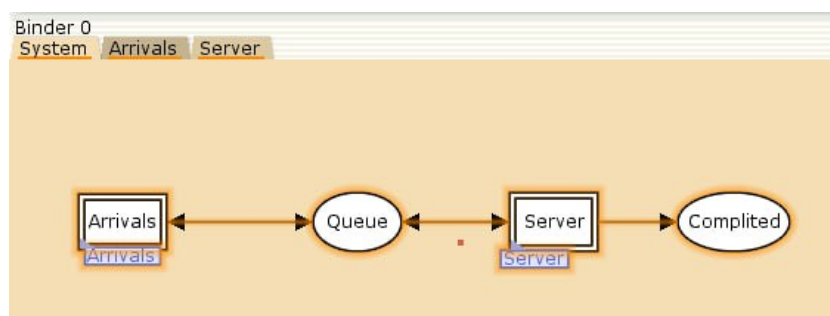


Рис. 3.1: Граф сети системы обработки заявок в очереди

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).

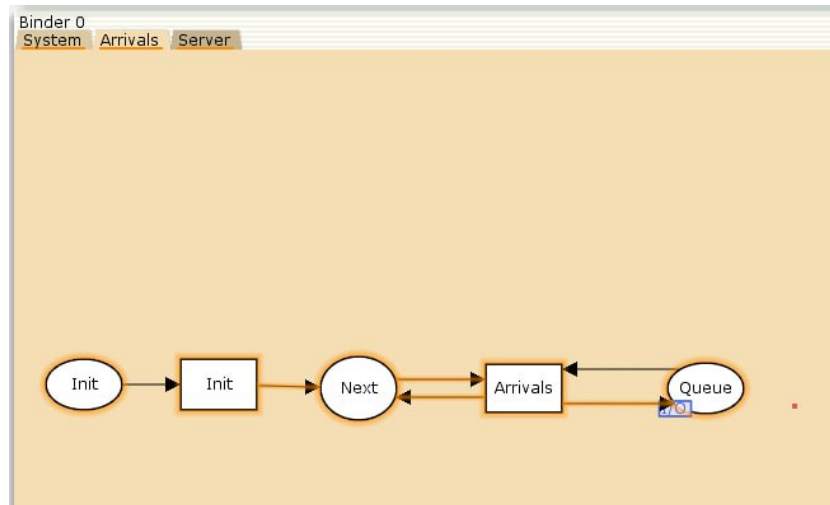


Рис. 3.2: Граф генератора заявок системы

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

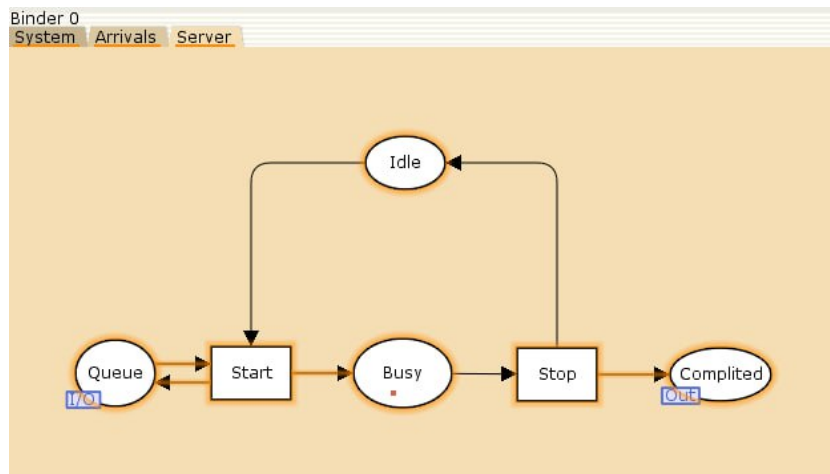


Рис. 3.3: Граф процесса обработки заявок на сервере системы

Зададим декларации системы. 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 — A и B; - 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. var proctime : INT; var job: Job; var jobs: Jobs;

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)

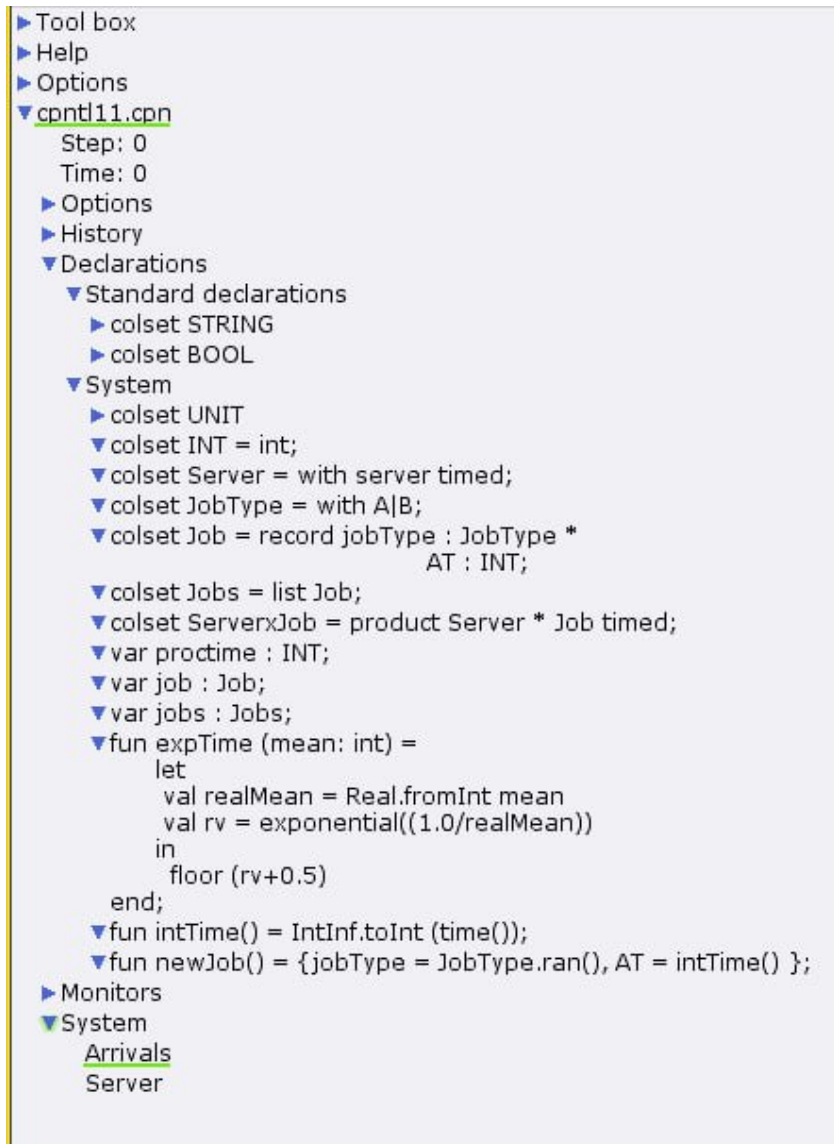


Рис. 3.4: Задание декларации системы

Зададим параметры модели на графах сети. На листе System : - у позиции Queue множество цветов фишек — Jobs; начальная маркировка 1[] определяет, что изначально очередь пуста. - у позиции Completed множество цветов фишек — Job.

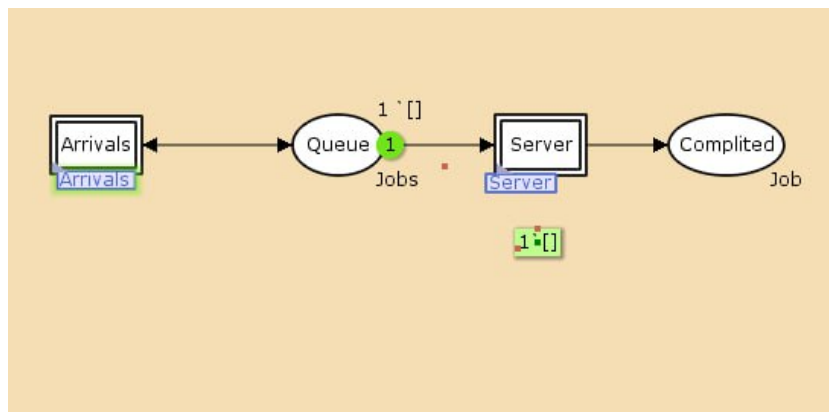


Рис. 3.5: Параметры элементов основного графа системы обработки заявок в очереди

На листе Arrivals : - 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.

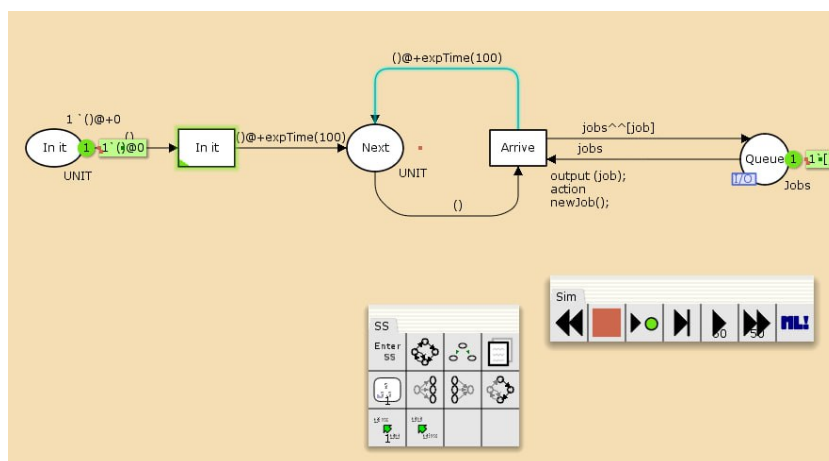


Рис. 3.6: Параметры элементов генератора заявок системы

На листе Server : - 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.

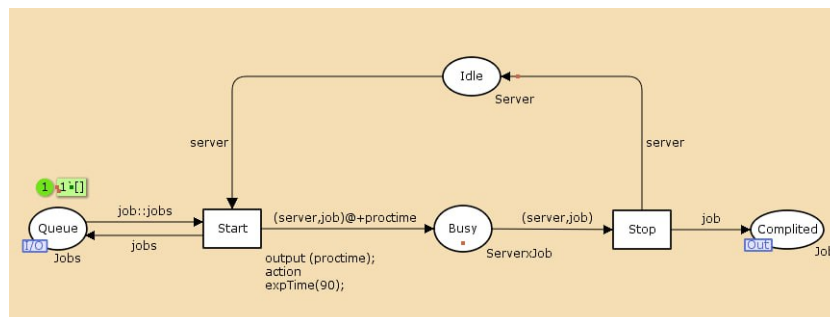


Рис. 3.7: Параметры элементов обработчика заявок системы

Мониторинг параметров очереди системы M|M|1. 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`

```

Binder 0
System Arrivals Server fun pred<Остановка> fun expTime colset Job fun obs
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 Остановка

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

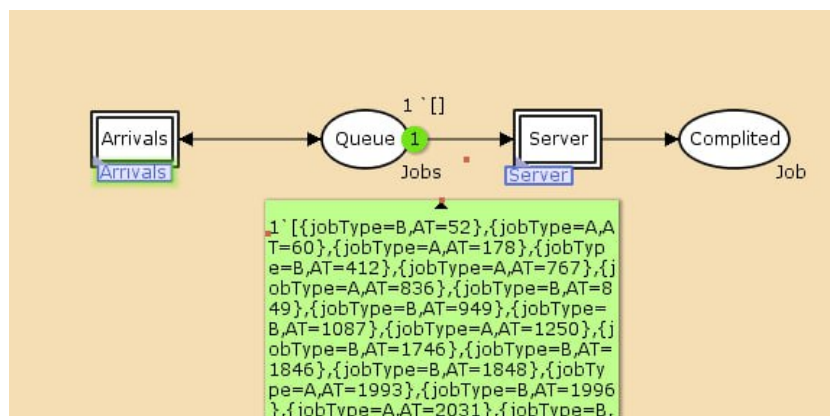
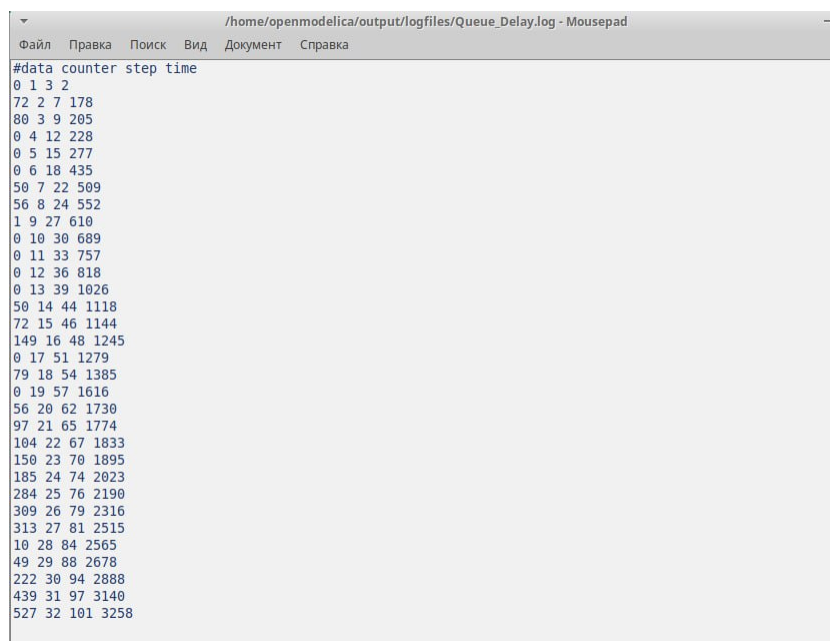


Рис. 3.9: Запуск системы обработки заявок в очереди

файл Queue_Delay.log, содержащий в первой колонке — значение задержки очереди, во второй — счётчик, в третьей — шаг, в четвёртой — время.



```
#data counter step time
0 1 3 2
72 2 7 178
80 3 9 205
0 4 12 228
0 5 15 277
0 6 18 435
50 7 22 509
56 8 24 552
1 9 27 610
0 10 30 689
0 11 33 757
0 12 36 818
0 13 39 1026
50 14 44 1118
72 15 46 1144
149 16 48 1245
0 17 51 1279
79 18 54 1385
0 19 57 1616
56 20 62 1730
97 21 65 1774
104 22 67 1833
150 23 70 1895
185 24 74 2023
284 25 76 2190
309 26 79 2316
313 27 81 2515
10 28 84 2565
49 29 88 2678
222 30 94 2888
439 31 97 3140
527 32 101 3258
```

Рис. 3.13: Файл Queue_Delay.log

С помощью gnuplot можно построить график значений задержки в очереди (рис. 11.10), выбрав по оси x время, а по оси y — значения задержки:

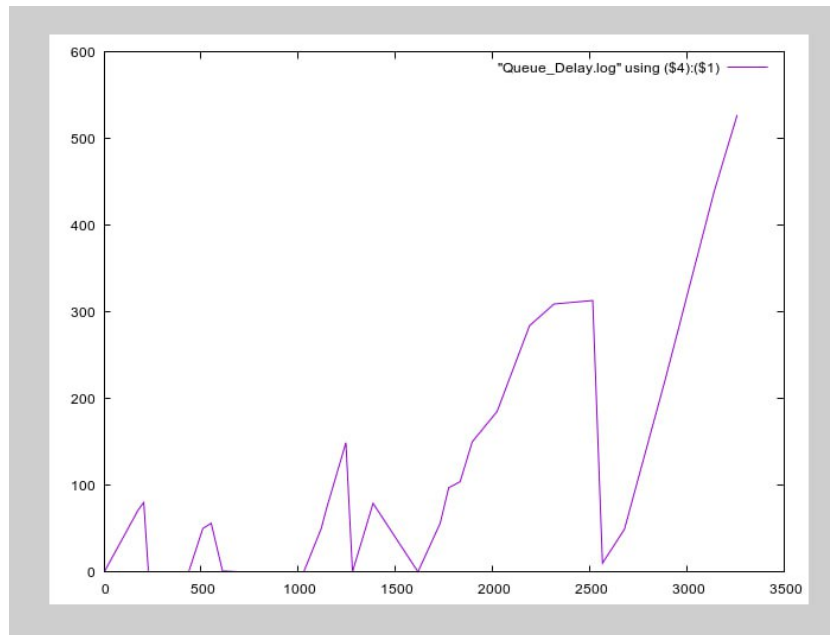


Рис. 3.14: График изменения задержки в очереди

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

```

server 1
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
end

```

Рис. 3.15: Функция Observer монитора Queue Delay Real

По сравнению с прошлым 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.

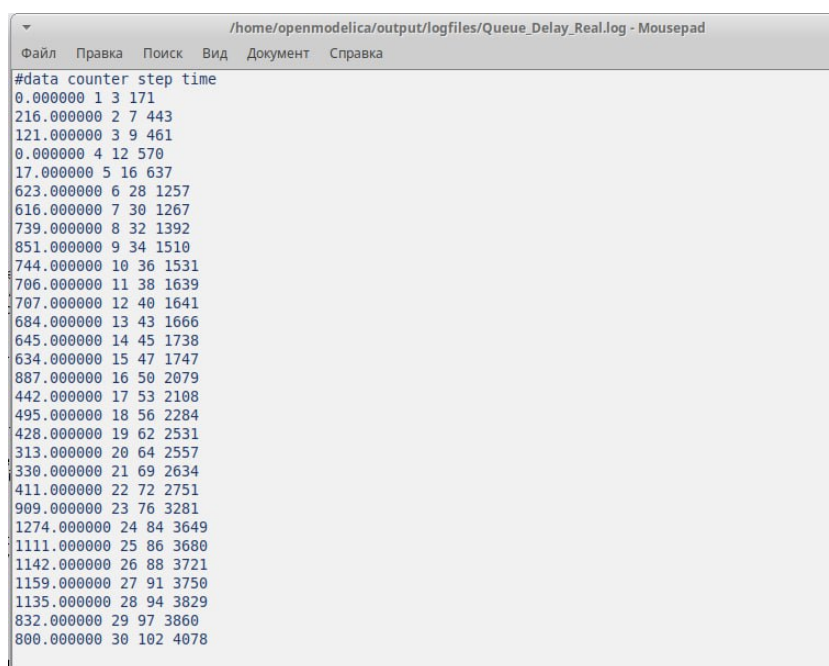


Рис. 3.16: Файл Queue_Delay_Real.log

Посчитаем, сколько раз задержка превысила заданное значение. С помощью палитры Monitoring выбираем Data Call и устанавливаем на переходе Start. Монитор называем Long Delay Time.

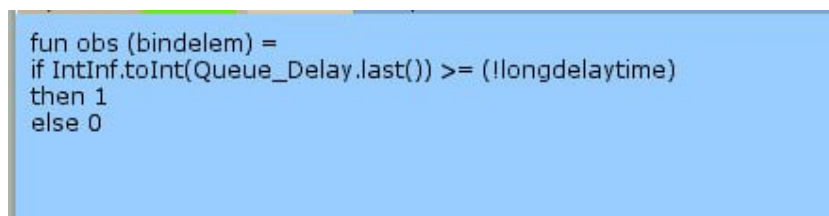


Рис. 3.17: Функция Observer монитора Long Delay Time

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

При этом необходимо в декларациях (рис. 11.13) задать глобальную переменную (в форме ссылки на число 200)

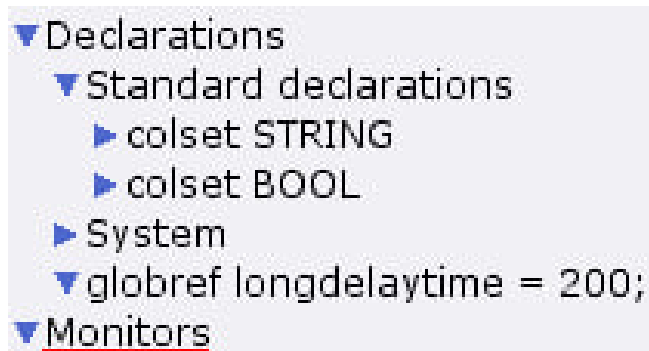


Рис. 3.18: Функция Observer монитора Long Delay Time

После запуска программы на выполнение в каталоге с кодом программы появится файл Long_Delay_Time.log

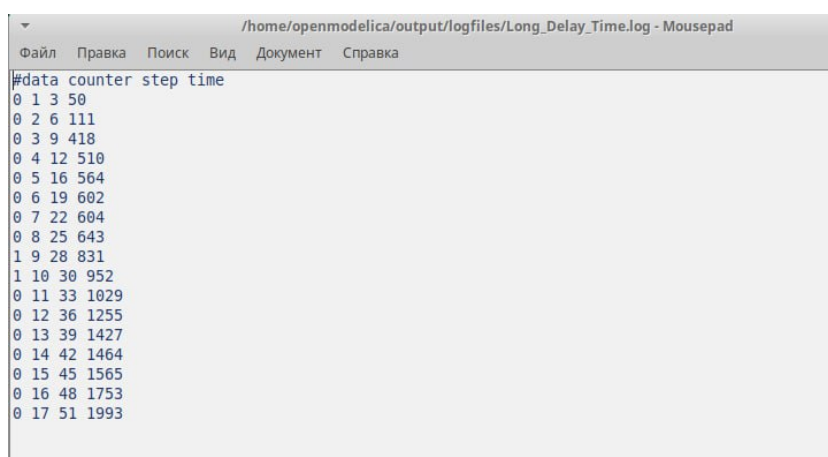


Рис. 3.19: Файл Long_Delay_Time.log

С помощью gnuplot можно построить график , демонстрирующий, в какие периоды времени значения задержки в очереди превышали заданное значение 200

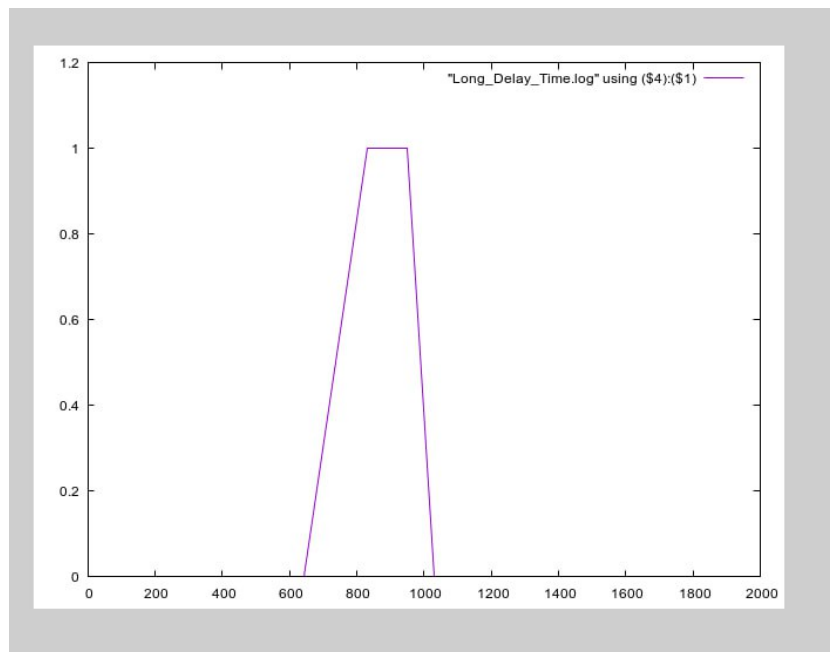


Рис. 3.20: Периоды времени, когда значения задержки в очереди превышали заданное значение

4 Выводы

В процессе лабораторной работы мы реализовали модель системы массового обслуживания $M|M|1$ в CPNTools