

СМО М/М/1.

Лабораторная работа №11.

Рогожина Н.А.

19 апреля 2025

Российский университет дружбы народов, Москва, Россия

Информация

- Рогожина Надежда Александровна
- студентка 3 курса НФИбд-02-22
- Российский университет дружбы народов
- <https://mikogreen.github.io/>

Задание

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.

Выполнение лабораторной работы

Первым делом опишем граф (на **Arrivals** и **Server** сделаем иерархию) системы и почти все необходимые декларации.

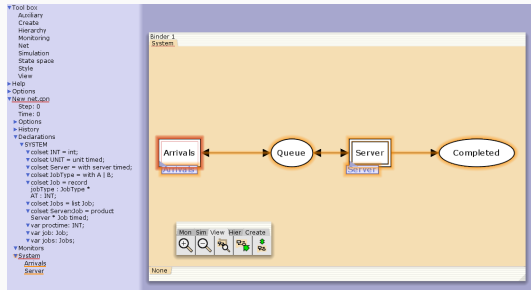


Рис. 1: Граф системы и декларации

Далее, зададим необходимые функции.

```
▼ 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()};
```

Рис. 2: Функции

Опишем граф генератора заявок системы (Arrivals).

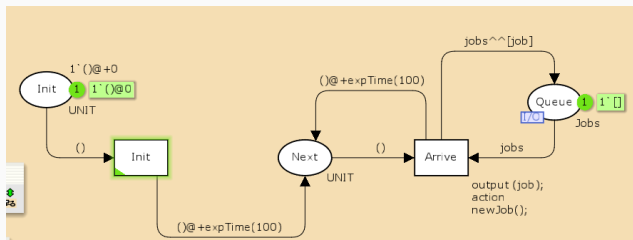


Рис. 3: Генератор заявок

И также опишем граф обработки заявок на сервере системы (**Server**).

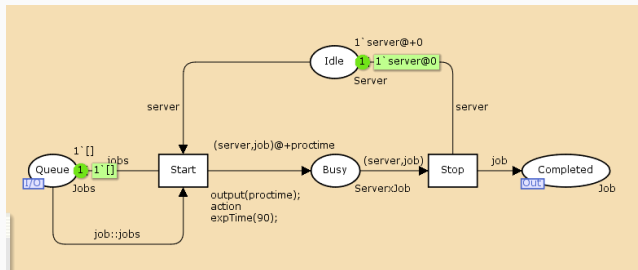


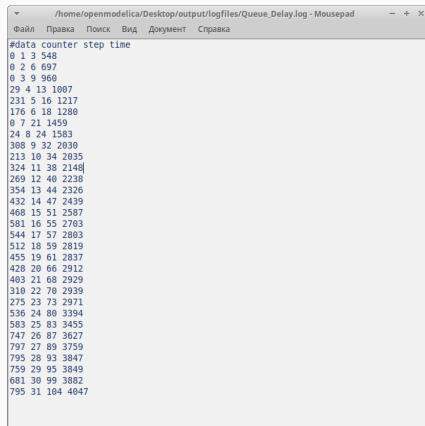
Рис. 4: Обработка заявок

Также, нам необходимо реализовать (последовательно) 4 мониторинга системы. Первые два - Queue_Delay и Ostanovka.

```
▼ Monitors
  ▼ 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
        let
          Init function
          Stop
        end
    ▼ 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
          end
    ▼ System
```

Рис. 5: Queue_Delay, Ostanovka

Запустив около 100 шагов моделирования, мы получили текстовый лог Queue_Delay.log.



```
#data counter step time
0 1 3 548
0 2 6 697
0 3 9 960
29 4 13 1007
231 5 16 1217
176 6 18 1280
0 7 21 1459
24 8 24 1583
308 9 32 2030
213 10 34 2035
324 11 38 2148
269 12 40 2238
354 13 44 2326
432 14 47 2439
468 15 51 2587
581 16 55 2703
544 17 57 2803
512 18 59 2819
455 19 61 2837
428 20 66 2912
403 21 68 2929
310 22 70 2939
275 23 73 2971
536 24 80 3394
583 25 83 3455
747 26 87 3627
797 27 89 3759
795 28 93 3847
759 29 95 3849
681 30 99 3882
795 31 104 4047
```

Рис. 6: Результат

С помощью GNUplot мы его визуализировали и получили следующий график.

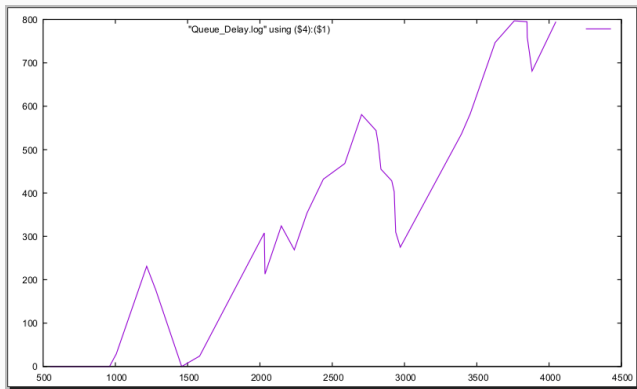


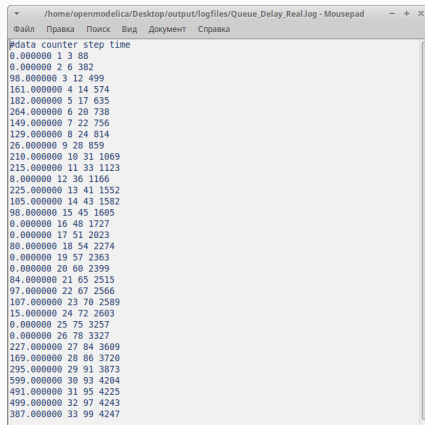
Рис. 7: Queue_Delay

Третий мониторинг системы это `Queue_Delay_Real`. Он повторяет `Queue_Delay`, только в действительных значениях.

```
▼ 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
      let
        ► Init function
        ► Stop
        ► Ostanovka
```

Рис. 8: Queue_Delay_Real

Сбросив моделирование до начальной точки и запустив заново, получили задержку в действительных числах.



```

#data counter step time
0.000000 1 3 88
0.000000 2 6 382
98.000000 3 12 499
161.000000 4 14 574
182.000000 5 17 635
264.000000 6 20 738
149.000000 7 22 756
129.000000 8 24 814
26.000000 9 28 859
210.000000 10 31 1069
215.000000 11 33 1123
8.000000 12 36 1166
225.000000 13 41 1552
105.000000 14 43 1582
98.000000 15 45 1605
0.000000 16 48 1727
0.000000 17 51 2023
80.000000 18 54 2274
0.000000 19 57 2363
0.000000 20 60 2399
84.000000 21 65 2515
97.000000 22 67 2566
107.000000 23 70 2589
15.000000 24 72 2603
0.000000 25 75 3257
0.000000 26 78 3327
227.000000 27 84 3609
169.000000 28 86 3720
295.000000 29 91 3873
599.000000 30 93 4204
491.000000 31 95 4225
499.000000 32 97 4243
387.000000 33 99 4247
    
```

Рис. 9: Queue_Delay_Real.log

С помощью GNUplot мы его визуализировали и получили следующий график.

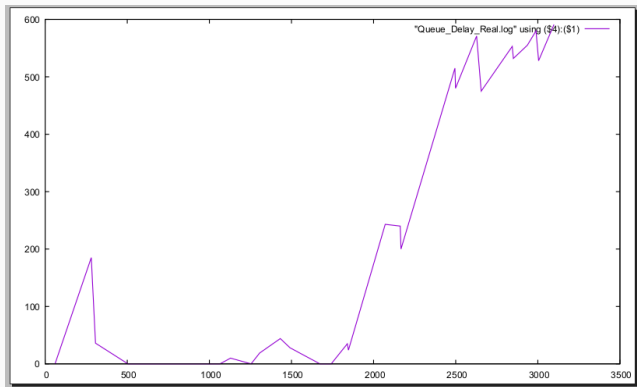


Рис. 10: Queue_Delay_Real

Последний мониторинг, который было необходимо реализовать - `Long_Delay_Time`. Здесь будем считать, сколько раз задержка превысила заданное значение. Также было необходимо задать еще одну декларацию - `longdelaytime`.

```
▼ Declarations
  ▼ globref longdelaytime = 200;
  ► SYSTEM
▼ 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. 11: Long_Delay_Time

Повторив обнуление и 100 шагов, получили результат симуляции Long_Delay_Time.log.

```

/home/openmodelica/Desktop/output/logfiles/Long_Delay_Time.l - + x
Файл  Правка  Поиск  Вид  Документ  Справка
#data counter step time
0 1 3 548
0 2 6 697
0 3 9 960
0 4 13 1007
1 5 16 1217
0 6 18 1280
0 7 21 1459
0 8 24 1583
1 9 32 2030
1 10 34 2035
1 11 38 2148
1 12 40 2238
1 13 44 2326
1 14 47 2439
1 15 51 2587
1 16 55 2703
1 17 57 2803
1 18 59 2819
1 19 61 2837
1 20 66 2912
1 21 68 2929
1 22 70 2939
1 23 73 2971
1 24 80 3394
1 25 83 3455
1 26 87 3627
1 27 89 3759
1 28 93 3847
1 29 95 3849
1 30 99 3882

```

Визуализируем количество превышений.

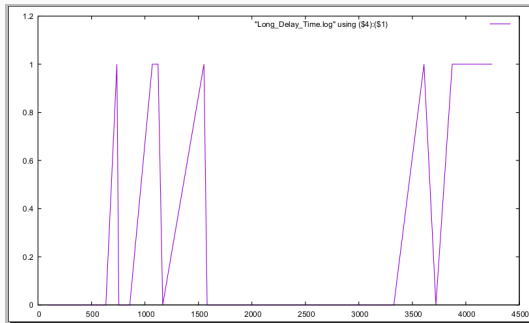


Рис. 13: Long_Delay_Time

Выводы

В ходе лабораторной работы мы смоделировали поведение СМО М/М/1 с помощью CpnTools.