

Лабораторная работа № 11

Модель системы массового обслуживания $M|M|1$

Мугари Абдеррахим

19 апреля 2025

Российский университет дружбы народов, Москва, Россия

Информация

- Анна Владиславовна Королькова
- доцент кафедры прикладной информатики и теории вероятностей РУДН;
- заведующий лабораторией кафедры прикладной информатики и теории вероятностей РУДН (по совместительству);
- программист I кат.
- Российский университет дружбы народов
- korolkova-av@rudn.ru

- Мугари Абдеррахим
- Студент третьего курса
- фундаментальная информатика и информационные технологии
- Российский университет дружбы народов
- 1032215692@rudn.ru
- <https://iragoum.github.io/>



Цель работы

Создание модели системы массового обслуживания $M|M|1$ с использованием среды моделирования CPN Tools.

- Построить модель $M|M|1$ в CPN Tools.
- Настроить мониторинг ключевых параметров работы системы.
-

Выполнение лабораторной работы

Необходимо смоделировать систему, в которую поступают заявки двух типов. Поток заявок подчиняется пуассоновскому распределению. Все заявки направляются в очередь на обработку, где применяется дисциплина обслуживания FIFO (первым пришёл — первым обслужен). Если сервер свободен, он немедленно начинает обработку поступившей заявки

Модель реализована в среде CPN Tools и разделена на три отдельных листа:

1. **System** — основная схема модели.
2. **Generator** — схема генерации заявок.
3. **Server** — схема обработки заявок .

- Основной граф модели (лист System)

На листе описывается структура системы:

- **Позиции:**
 - **Queue** — очередь заявок;
 - **Completed** — завершённые заявки.

- Переходы:
 - **Arrivals** — генерация новых заявок;
 - **Server** — передача заявки на обработку.

Оба перехода имеют иерархическую структуру, которую можно настроить с помощью инструмента *Hierarchy*.

Между **Arrivals** и **Queue**, а также между **Queue** и **Server** — двусторонняя связь. Между **Server** и **Completed** — односторонняя.

- Граф генерации заявок (лист Generator)

Здесь реализована логика поступления заявок:

- **Позиции:**
 - `Init` — текущая заявка;
 - `Next` — следующая заявка;
 - `Queue` — ссылка на позицию очереди из листа *System*.
- **Переходы:**
 - `Init` — моделирует поступление заявок с экспоненциальным распределением (интенсивность: 100 заявок в единицу времени);
 - `Arrive` — подача заявки в очередь.

- Граф обработки заявок (лист Server)

Этот лист описывает поведение сервера:

- **Позиции:**
 - **Busy** — сервер в работе;
 - **Idle** — сервер свободен;
 - **Queue** и **Complited** — позиции из листа *System*.
- **Переходы:**
 - **Start** — начало обработки заявки;
 - **Stop** — завершение обработки.

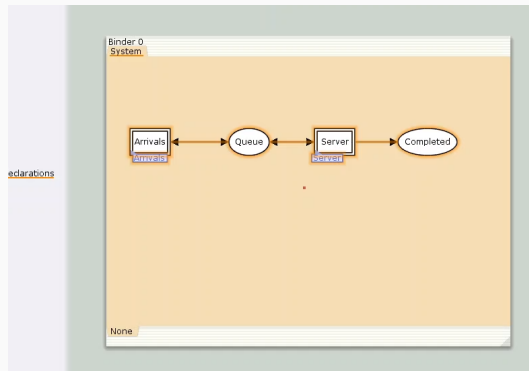


Рис. 1: Граф сети системы обработки заявок в очередь

Граф генератора заявок системы

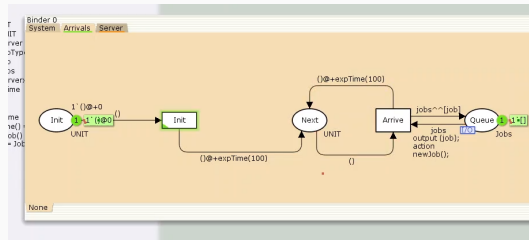


Рис. 2: Граф генератора заявок системы

Граф процесса обработки заявок на сервере системы

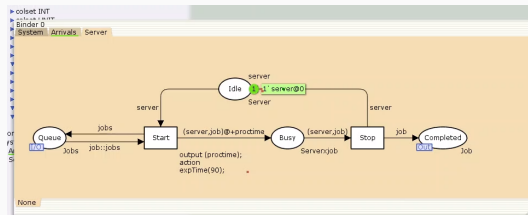


Рис. 3: Граф процесса обработки заявок на сервере системы

В дальнейшем необходимо определить декларации для каждой части модели

- Множества цветов (colorset)

В модели определены следующие типы данных (множества цветов), используемые для представления различных сущностей в системе:

- **UNIT** — используется для фиксации моментов времени.
- **INT** — обозначает моменты поступления заявок в систему.
- **JobType** — определяет два возможных типа заявок: **A** и **B**.
- **Job** — кортеж, состоящий из двух полей:
 - **jobType** (тип **JobType**) — обозначает тип заявки;
 - **AT** (тип **INT**) — хранит время, в течение которого заявка находится в системе.
- **Jobs** — список заявок (тип — список объектов **Job**).
- **ServerxJob** — состояние сервера, когда он занят конкретной заявкой.

Модель использует следующие переменные:

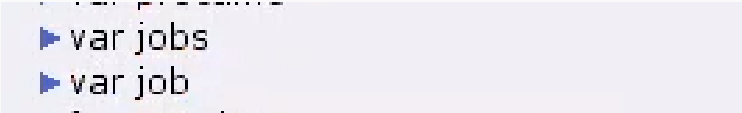
- `proctime` — время, затрачиваемое на обработку одной заявки;
- `job` — отдельная заявка;
- `jobs` — список заявок, поступивших в очередь.

Для корректной генерации и обработки заявок определены следующие функции:

- `expTime()` — возвращает случайные значения, моделирующие интервалы времени между поступлениями заявок на основе экспоненциального распределения;
- `intTime()` — преобразует текущее модельное время в целое число;
- `newJob()` — создает новую заявку (**Job**), случайным образом выбирая тип (А или В).

```
me: U
ptions
istory
eclarations
SYSTEM
  ▶ colset INT
  ▶ colset UNIT
  ▶ colset Server
  ▶ colset JobType
  ▶ colset Job
  ▶ colset Jobs
  ▼ colset Serverxjob = product Server * Job timed;
```

Рис. 4: Определения множества цветов системы



▶ var jobs
▶ var job

Рис. 5: Определение переменных модели

```
▶ fun expTime  
▼ fun intTime() = IntInf.toInt(time());  
▼ fun newJob() = {  
    jobType = JobType.ran() , AT = intTime()};
```

Рис. 6: Определение функций системы

Для корректной работы модели в CPN Tools были заданы параметры элементов на всех трех листах: **System**, **Arrivals** и **Server**.

На данном листе представлены основные элементы сети массового обслуживания:

- **Позиция Queue:**
 - Цветовое множество: **Jobs**.
 - Начальная маркировка: **1' []**, что указывает на пустую очередь в начале моделирования.
- **Позиция Completed:**
 - Цветовое множество: **Job**.

Параметры элементов основного графа системы обработки заявок в очереди

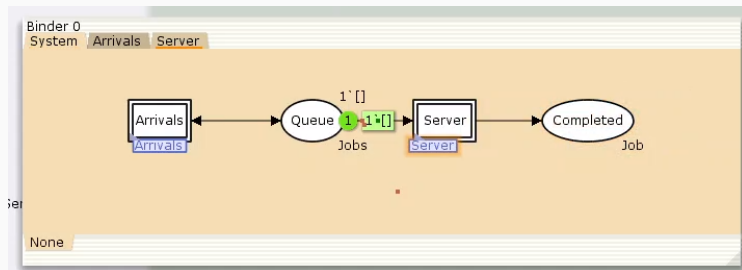


Рис. 7: Параметры элементов основного графа системы обработки заявок в очереди

На этом листе формируются заявки, поступающие в систему:

- **Позиция Init:**
 - Цветовое множество: `UNIT`.
 - Начальная маркировка: `1'()@0`, что означает начало генерации заявок с нулевого времени.
- **Позиция Next:**
 - Цветовое множество: `UNIT`.
- **Переход Init:**
 - На дуге из `Init`: выражение `()` — иницирует генерацию заявок.
 - На дугах от `Init` и `Arrive` к `Next`: `()@+expTime(100)` — задаёт интервалы между заявками по экспоненциальному распределению со средней интенсивностью 100.
- **Переход Arrive:**
 - На дуге из `Next`: `()` — передаёт сигнал генерации.
 - На дуге к `Queue`: `jobs^[job]` — добавляет заявку в очередь.
 - Обратная связь с `Queue`: `jobs`.

Параметры элементов генератора заявок системы

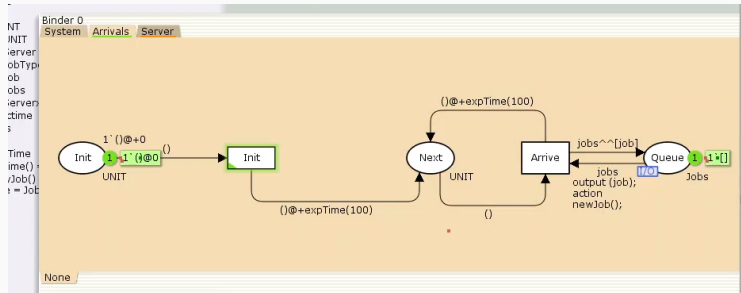


Рис. 8: Параметры элементов генератора заявок системы

На этом листе смоделирован сервер, обрабатывающий заявки:

- **Позиция Busy:**
 - Цветовое множество: `Server`.
 - Начальная маркировка: `1'server@+0` — сервер свободен в начале моделирования.
- **Позиция Idle:**
 - Цветовое множество: `ServerxJob`.
- **Переход Start:**
 - Output: `proctime`.
 - Action: `expTime(90)` — определяет, что время обработки заявки подчиняется экспоненциальному распределению со средним значением 90.

- На дуге от `Queue: job::jobs` — позволяет начать обработку, если есть заявки.
- К `Busy: (server, job)@+proctime` — передача заявки на сервер с учетом времени обработки.
- Обратная связь в `Queue: jobs`.
- Переход **Stop**:
 - От `Busy: (server, job)` — завершение обработки заявки.
 - К `Completed: job` — заявка считается обслуженной.
 - Состояние сервера обновляется через `Idle: server`.

Параметры элементов графа обработки заявок

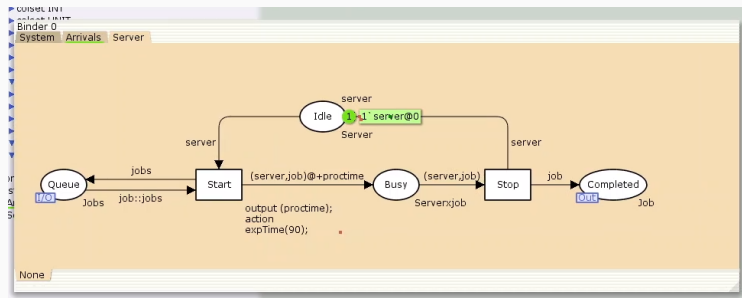


Рис. 9: Параметры элементов графа обработки заявок

Запуск системы обработки заявок в очереди

После задания всех необходимых параметров компоненты обработчика заявок активируются, и система начинает функционировать, как показано на рисунке ниже.

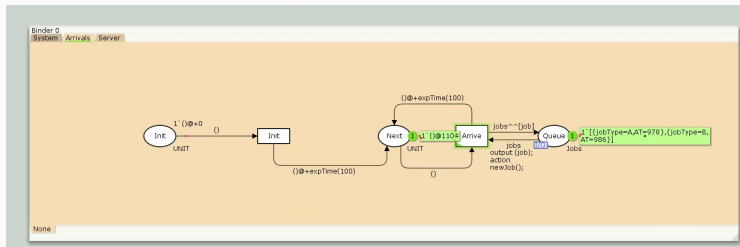


Рис. 10: Запуск системы обработки заявок в очереди

Для отслеживания параметров используется палитра **Monitoring**. В первую очередь добавляется **Break Point**, который размещается на переходе **Start**. После этого в меню **Monitor** появляется новый раздел, назовём его **Ostanovka**.

В этом разделе необходимо изменить функцию **Predicate**, отвечающую за условие активации монитора. Стандартное значение **true** заменяется на выражение `Queue_Delay.count()=200`, чтобы монитор срабатывал каждые 200 заявок.

Функция Predicate монитора Ostanovka

CPN Tools

▼ Tool box

- Auxiliary
- Create
- Hierarchy
- Monitoring
- Net
- Simulation
- State space
- Style
- View
- Help
- Options
- ▼ lab111.cpn
 - Step: 0
 - Time: 0
 - Options
 - History
 - ▼ Declarations
 - ▼ SYSTEM
 - colset INT
 - colset UNIT
 - colset Server
 - colset JobType
 - colset Job
 - colset Jobs
 - ▼ colset Server
 - fun obs (bindelem) =
 - let
 - fun obsBindElem (ServerStart (1, {job,jobs,proctime})) = |
 - obsBindElem _ = ~1
 - in
 - obsBindElem bindelem
 - end
 - var proctime
 - var jobs
 - var job
 - fun expTime
 - ▼ fun intTime() = IntInf.toInt(time());
 - ▼ fun newJob() = {
 - jobType = JobType.ran() , AT = intTime();
 - ▼ Monitors
 - ▼ Queue Delay
 - Type: Data collection
 - Nodes ordered by pages
 - Predicate
 - ▼ Observer
 - fun obs (bindelem) =
 - let
 - fun obsBindElem (ServerStart (1, {job,jobs,proctime})) = 0
 - | obsBindElem _ = ~1
 - in
 - obsBindElem bindelem
 - end

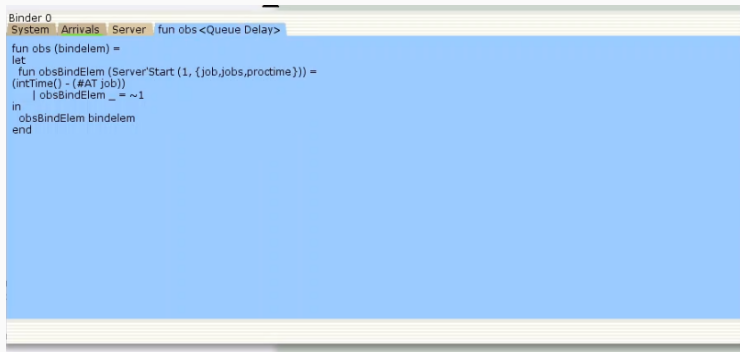
| View | Sim | Mc | Creal | Mon |
|-----------|-----------|-------------|-----------|---------------|
| Data Coll | Mark Size | Break point | User def | Write in file |
| LL DC | Coun Tran | Place Cont | Tran Enab | |

Определение функции `Queue_Delay.count()`

С помощью палитры **Monitoring** выбирается элемент **Data Call**, который размещается также на переходе **Start**. Новый монитор следует назвать **Queue Delay** (без подчеркивания).

Функция **Observer** будет выполняться, когда предикат возвращает **true**. По умолчанию она возвращает 0 или ~1, где подчёркивание обозначает произвольный аргумент. Чтобы получить значение задержки, нужно из текущего времени `intTime()` вычесть временную метку **AT**, которая указывает момент поступления заявки в очередь.

Функция Observer монитора Queue Delay



```
Binder 0
System Arrivals Server fun obs<Queue Delay>
fun obs (bindelem) =
let
  fun obsBindElem (Server'Start (1, {job,jobs,proctime})) =
    (intTime() - (#AT job))
    | obsBindElem _ = ~1
in
  obsBindElem bindelem
end
```

Рис. 12: Функция Observer монитора Queue Delay

Вычисление задержки в действительных значениях

Для получения задержки в виде действительных чисел, снова используется **Data Call** на переходе **Start**. Новый монитор называется **Queue Delay Real**. Функцию **Observer** следует изменить так, чтобы результат преобразовывался в тип **real** (например, используя **~1.0**).

После запуска системы в каталоге проекта появится файл `Queue_Delay_Real.log` с данными, аналогичными файлу `Queue_Delay.log`, но с действительными значениями.

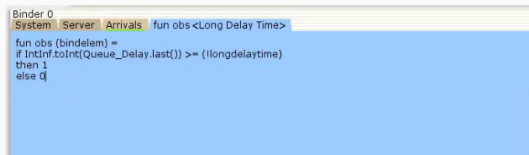
```

10 binder 0
11 System Arrivals Server fun obs.cQueue.Delay.Real()
12
13 fun obs (bindelem) =
14   let
15     fun obsBindElem (ServerStart (1, {job,jobs,proctime})) =
16       Real.fromInt(intTime() - (#AT job))
17       | obsBindElem _ = ~1.0
18   in
19     obsBindElem bindelem
20   end
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Рис. 13: Функция Observer монитора Queue Delay Real

Чтобы посчитать количество случаев, когда задержка превысила заданное значение, снова используется **Data Call** на переходе **Start**. Новый монитор следует назвать **Long Delay Time**, и изменить в нём функцию **Observer**, как показано ниже.



```
Binder 0
System Server Arrivals fun obs<Long Delay Time>
fun obs (bindelem) =
  if IntInf.toInt(Queue_Delay.last()) >= (!longdelaytime)
  then 1
  else 0
```

Рис. 14: Функция Observer монитора Long Delay Time

Подсчёт случаев превышения задержки

После запуска системы создается файл `Queue_Delay.log`, где: - первая колонка — значение задержки, - вторая — счётчик, - третья — шаг, - четвёртая — время.

С помощью **gnuplot** можно построить график изменения задержки, используя: - по оси X — время, - по оси Y — значения задержки.

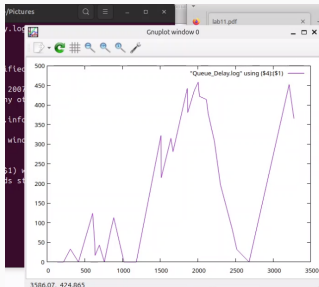


Рис. 15: График изменения задержки в очереди

Подсчёт случаев превышения задержки

С помощью **gnuplot** можно построить график, иллюстрирующий периоды времени, в которые значения задержки в очереди превышали установленный порог — 200. Этот график помогает визуализировать моменты перегрузки в системе (см. рисунок ниже).

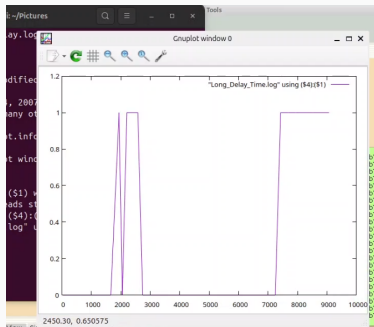


Рис. 16: Периоды времени, когда значения задержки в очереди превышали заданное значение

Выводы

В рамках выполненной работы была разработана и реализована модель системы массового обслуживания $M|M|1$ с использованием среды **CPN Tools**.