

# **Лабораторная работа 11**

Тагиев Байрам Алтай оглы

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>3</b>	<b>Выводы</b>	<b>11</b>

# Список иллюстраций

2.1	Граф сети модели . . . . .	6
2.2	Граф Arrivals . . . . .	6
2.3	Граф Sevrer . . . . .	6
2.4	Декларации модели . . . . .	7
2.5	Моделирование . . . . .	7
2.6	Функция Predicate монитора Ostanovka . . . . .	7
2.7	Функция Observer монитора Queue Delay . . . . .	8
2.8	График изменения задержки в очереди . . . . .	8
2.9	Функция Observer монитора Queue Delay Real . . . . .	9
2.10	График изменения задержки в очереди . . . . .	9
2.11	График изменения задержки в очереди . . . . .	10

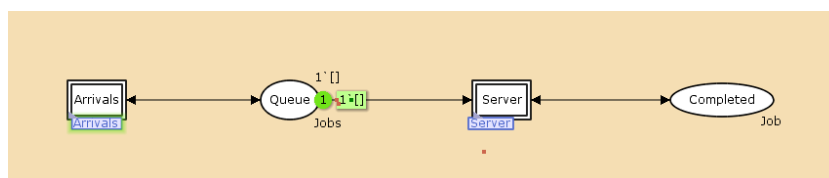
## **Список таблиц**

# 1 Цель работы

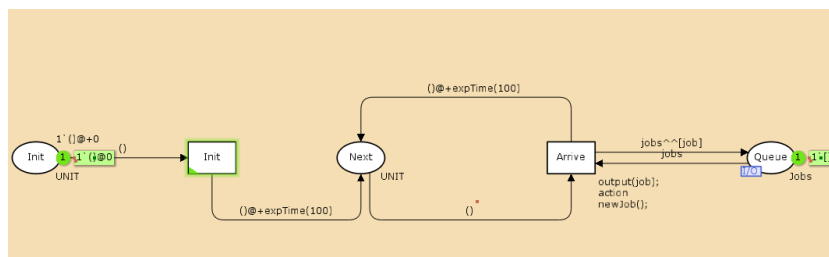
В систему поступает поток заявок двух типов, распределённый по пуассоновскому закону. Заявки поступают в очередь сервера на обработку. Дисциплина очереди - FIFO. Если сервер находится в режиме ожидания (нет заявок на сервере), то заявка поступает на обработку сервером.

## 2 Выполнение лабораторной работы

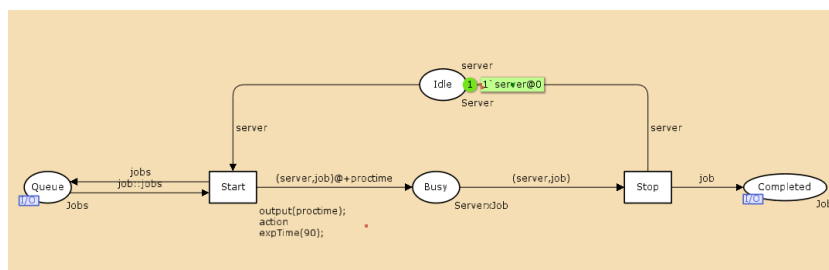
1. Рисуем граф сети.



Граф сети модели



Граф Arrivals



Граф Sevrer

## 2. Зададим декларации модель.

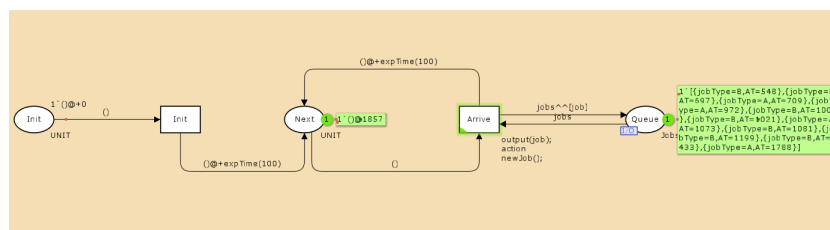
```

▼ colset UNIT = 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
▼ fun intTime () = IntInf.toInt (time());
▼ fun newJob () = {jobType = JobType.ran(),
                    AT = intTime()}

```

### Декларации модели

## 3. Если прокрутить моделирование, то сможешь увидеть как пакеты поступают в систему и обрабатываются.



### Моделирование

## 4. Добавим мониторы. Изменим предикат, задав число шагов, через которое будем останавливать мониторинг.

```

fun pred (bindelem) =
let
  fun predBindElem (ServerStart (1, {job,jobs,proctime})) = Queue_Delay.count()=200
  | predBindElem _ = false
in
  predBindElem bindelem
end

```

### Функция Predicate монитора Ostanovka

5. Добавим Data call.

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

Функция Observer монитора Queue Delay

6. Запустив, мы получим log файл, при помощи которого мы можем построить график изменения задержки в очереди.

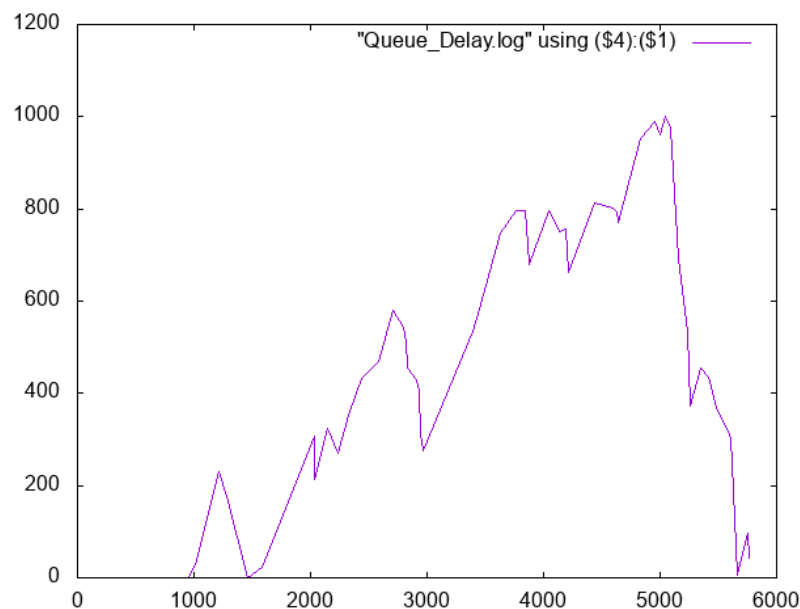


График изменения задержки в очереди

7. 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.



```

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

```

Функция Observer монитора Queue Delay Real

8. Запустив, мы получим log файл, при помощи которого мы можем построить график изменения задержки в очереди.

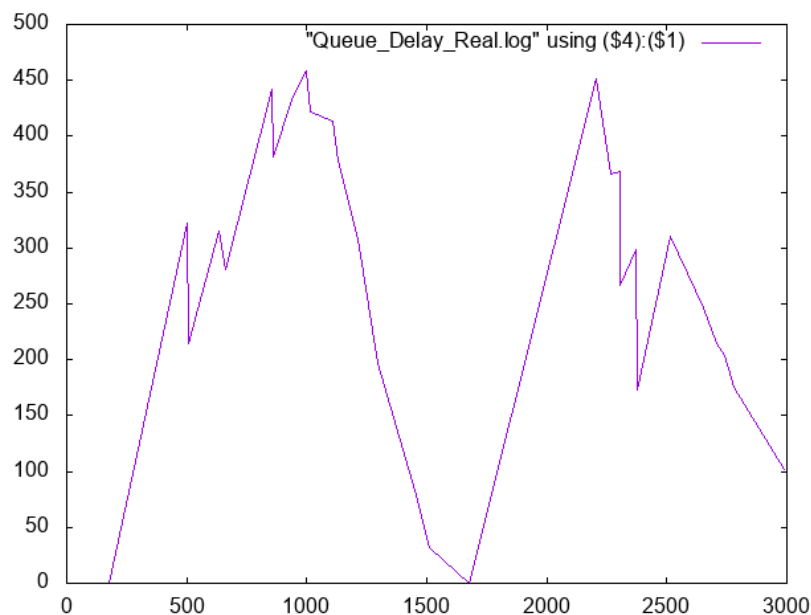


График изменения задержки в очереди

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.
10. Запустив, мы получим log файл, при помощи которого мы можем построить график изменения задержки в очереди.

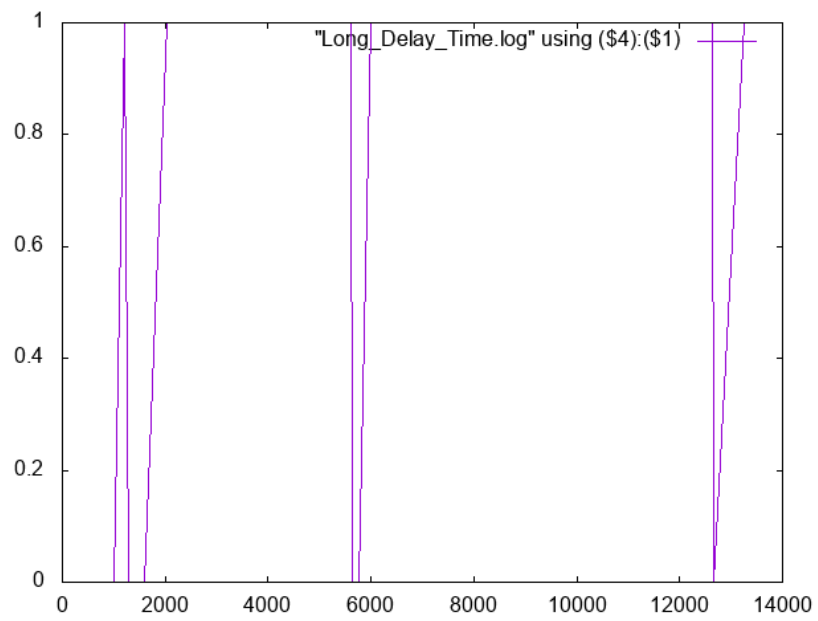


График изменения задержки в очереди

## 3 Выводы

Во время выполнения лабораторной работы, я провел моделирование  $M|M|1$ .