



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени
Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ «Информатика и системы управления»
КАФЕДРА _____ «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ
по лабораторной работе №5
по курсу «Моделирование»
на тему: «Информационный центр»

Студент	<u>ИУ7-73Б</u>	_____	<u>Лагутин Д. В.</u>
	(Группа)	(Подпись, дата)	(Фамилия И. О.)
Преподаватель		_____	<u>Рудаков И. В.</u>
		(Подпись, дата)	(Фамилия И. О.)

Цель работы

В информационный центр приходят клиенты через интервал времени 10 ± 2 минуты. Если все три имеющихся оператора заняты, клиенту отказывают в обслуживании. Операторы имеют разную производительность и могут обеспечивать обслуживание среднего запроса пользователя за 20 ± 5 ; 40 ± 10 ; 40 ± 20 . Клиенты стремятся занять свободного оператора с максимальной производительностью. Полученные запросы сдаются в накопитель. Откуда выбираются на обработку. На первый компьютер запросы от 1 и 2-ого операторов, на второй – запросы от 3-его. Время обработки запросов первым и 2-м компьютером равны соответственно 15 и 30 мин. Промоделировать процесс обработки 300 запросов. Найти вероятность отказа.

Моделируемая система

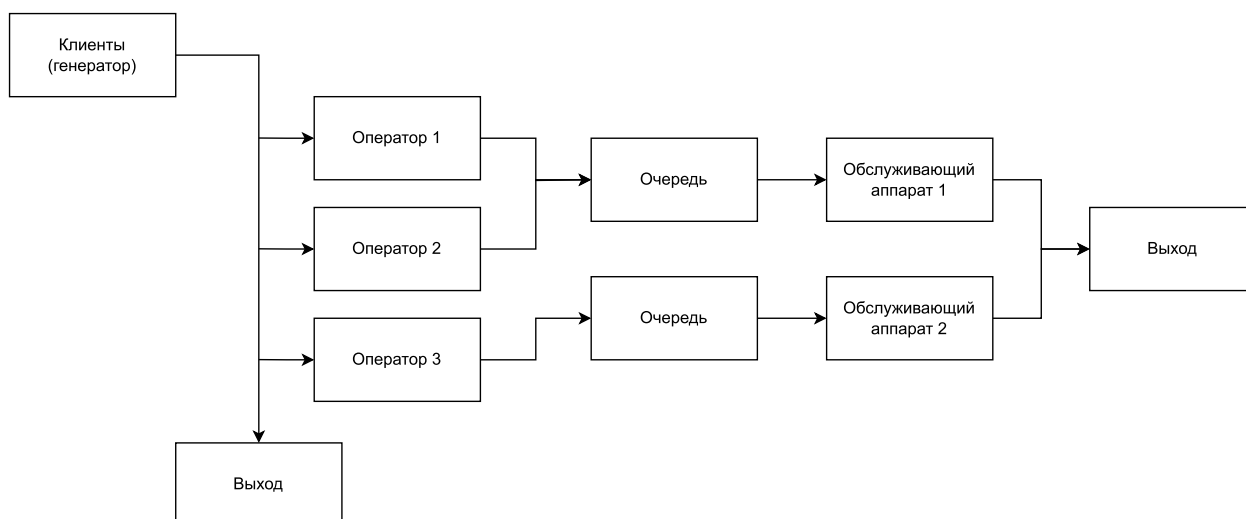


Рисунок 1 – Схема моделируемой системы

В процессе взаимодействия клиентов с информационным центром возможно:

- 1) режим нормального обслуживания, т.е. клиент выбирает одного из свободных операторов, отдавая предпочтение тому у которого меньше

номер;

- 2) режим отказа в обслуживании клиента, когда все операторы заняты.
Вероятность отказа находится по следующей формуле:

$$P_{\text{отказ}} = \frac{N_{\text{отказ}}}{N_{\text{всего}}}$$

Текст программы

Листинг 1 – Реализация элементов системы

```
1 class Model
2 {
3     public:
4         Model(std::string name);
5         virtual ~Model(void) = 0;
6
7         virtual const std::string &getName(void) const;
8
9         virtual void
10        setModifier(std::shared_ptr<RequestModifier>);
11        virtual std::shared_ptr<RequestModifier>
12        getModifier(void) const;
13
14    private:
15        const std::string name;
16        std::shared_ptr<RequestModifier> modifier;
17 };
18
19 class Request
20 {
21     public:
22         virtual ~Request(void) = 0;
23 };
24
25 class Sender;
26
27 class Receiver
```

```

28 {
29     public:
30         Receiver(std::shared_ptr<Pipe> pipe);
31         virtual ~Receiver(void) = 0;
32
33         void connectInPipe(std::shared_ptr<Pipe> pipe);
34         std::shared_ptr<Pipe> inpipe(void);
35         void askSender(void);
36
37         void registerSender(std::shared_ptr<Sender> sender);
38
39     private:
40         std::shared_ptr<Pipe> _inpipe;
41         std::list<std::shared_ptr<Sender>> _senders;
42 };
43
44 class Sender
45 {
46     public:
47         Sender(std::shared_ptr<Pipe> pipe);
48         virtual ~Sender(void) = 0;
49
50         void connectOutPipe(std::shared_ptr<Pipe> pipe);
51         std::shared_ptr<Pipe> outpipe(void);
52
53         virtual void callback(void) = 0;
54
55     private:
56         std::shared_ptr<Pipe> _outpipe;
57 };
58
59 class Runner
60 {
61     public:
62         virtual ~Runner(void) = default;
63         virtual void run(void) = 0;
64 };
65
66 class BasicPipe : public Pipe
67 {
68     public:

```

```

69         BasicPipe(std::string name);
70         virtual ~BasicPipe(void) override = default;
71
72         virtual bool empty(void) const override;
73         virtual bool push(std::shared_ptr<Request>) override;
74         virtual std::shared_ptr<Request> pop(void) override;
75
76         virtual void newSet(void) override;
77         virtual void dropCurrentSet(void) override;
78         virtual void clear(void) override;
79
80     private:
81         std::mutex mutex;
82         std::list<std::shared_ptr<Request>>> sets;
83 };
84
85 class Buffer : public Model, public Sender, public Receiver
86 {
87     public:
88         Buffer(std::string name, const size_t size,
89             std::shared_ptr<Pipe> inpipe,
90             std::shared_ptr<Pipe> outpipe);
91         virtual ~Buffer(void) override = default;
92
93         virtual size_t read(void);
94         virtual size_t send(size_t amount = 1);
95
96         virtual size_t used(void) const;
97
98         virtual void callback(void) override;
99
100    private:
101        const size_t size;
102        std::list<std::shared_ptr<Request>> memory;
103        std::mutex mutex;
104 };
105
106 class Gate : public Model, public Receiver
107 {
108     public:
109         class Out : public Model, public Sender

```

```

110     {
111         public:
112             Out(std::string name,
113                 std::shared_ptr<Pipe> outpipe);
114             virtual ~Out(void) override = default;
115
116             virtual bool send(std::shared_ptr<Request>);
117
118             virtual void callback(void) override;
119     };
120
121     public:
122         Gate(std::string name, std::shared_ptr<Pipe> inpipe,
123             std::list<std::shared_ptr<Pipe>> outpipes);
124         virtual ~Gate(void) override = default;
125
126         virtual void
127             setModifier(std::shared_ptr<RequestModifier>)
128             override;
129
130         virtual bool read(void);
131         virtual bool redirect(std::string name);
132         virtual const std::list<std::string> &list(void);
133
134     private:
135         std::list<std::shared_ptr<Out>> outs;
136         std::list<std::string> names;
137         std::shared_ptr<Request> current = nullptr;
138 };
139
140 class RequestCreator
141 {
142     public:
143         virtual ~RequestCreator(void) = default;
144         virtual std::shared_ptr<Request> create(void) = 0;
145 };
146
147 class Generator : public Model, public Sender
148 {
149     public:
150         Generator(std::string name,

```

```

151         std::shared_ptr<Pipe> outpipe = nullptr);
152     virtual ~Generator(void) override = default;
153     virtual void callback(void) override;
154
155     virtual void
156     setCreator(std::shared_ptr<RequestCreator> creator);
157
158     virtual void generate(const size_t amount = 1);
159
160     private:
161         std::shared_ptr<RequestCreator> cretator = nullptr;
162 };
163
164 class Processor : public Model, public Sender,
165                 public Receiver
166 {
167     public:
168         Processor(std::string name,
169                 std::shared_ptr<Pipe> inpipe = nullptr,
170                 std::shared_ptr<Pipe> outpipe = nullptr);
171         virtual ~Processor(void) override = default;
172
173         virtual bool read(void);
174         virtual bool isActive(void) const;
175         virtual bool release(void);
176
177         virtual void callback(void) override;
178
179     private:
180         std::shared_ptr<Request> active = nullptr;
181         std::list<std::shared_ptr<Request>> current;
182 };
183
184 class Terminator : public Model, public Receiver
185 {
186     public:
187         Terminator(std::string name,
188                 std::shared_ptr<Pipe> inpipe = nullptr);
189         virtual ~Terminator(void) override = default;
190
191         virtual void read(void);

```

```

192         std::list<std::shared_ptr<Request>> getDone(void);
193
194     private:
195         std::list<std::shared_ptr<Request>> done;
196 };
197
198 class StatatisticsBlock : public Model
199 {
200     public:
201         using ModelMap =
202             std::unordered_map<std::string,
203                             std::shared_ptr<Model>>;
204         class Strategy
205         {
206             public:
207                 virtual ~Strategy(void) = default;
208                 virtual void
209                     execute(const ModelMap &model) = 0;
210         };
211
212     public:
213         StatatisticsBlock(std::string name,
214                         std::list<std::shared_ptr<Model>>);
215         virtual ~StatatisticsBlock(void) override = default;
216
217         virtual void
218             registerStrategy(std::shared_ptr<Strategy> strategy);
219         virtual void write(void);
220
221     private:
222         std::list<std::shared_ptr<Strategy>> strategies;
223         ModelMap map;
224 };

```

Листинг 2 – Реализация подхода Δt

```

1 class TimeModel
2 {
3     public:
4         virtual ~TimeModel(void) = default;
5         virtual size_t priority(void) = 0;
6         virtual void tick(double time) = 0;

```



```

7         virtual void
8         setModifier(std::shared_ptr<RequestModifier>) = 0;
9     };
10
11     class TimeRunner : public Runner
12     {
13     public:
14         TimeRunner(size_t requests, double time, double step,
15                     std::shared_ptr<TimeRequestModifier>,
16                     std::list<std::shared_ptr<TimeModel>>>);
17         virtual ~TimeRunner(void) override = default;
18
19         virtual void run(void) override;
20
21     private:
22         const size_t requests;
23         const double end;
24         const double step;
25         std::shared_ptr<TimeRequestModifier> modifier;
26         std::map<size_t,
27                 std::list<std::shared_ptr<TimeModel>>> \
28             items;
29     };
30
31     TimeRunner::TimeRunner(size_t requests, double time,
32                             double step,
33                             std::shared_ptr<TimeRequestModifier>,
34                             std::list<std::shared_ptr<TimeModel>>>)
35     : requests(requests), end(std::abs(time)),
36       step(std::abs(step)), modifier(modifier)
37     {
38         if (nullptr == this->modifier)
39             throw std::logic_error("Nullptr modifier");
40
41         for (auto &item : items)
42         {
43             if (nullptr == item)
44                 throw std::logic_error("Nullptr modifier");
45
46             auto iter = this->items.find(item->priority());
47

```

```

48         if (this->items.end() == iter)
49             this->items.emplace(item->priority(),
50                                 std::list<std::shared_ptr<TimeModel>>({item}));
51         else
52             (*iter).second.push_back(item);
53     }
54 }
55
56 void TimeRunner::run(void)
57 {
58     double time = 0;
59     this->modifier->setTime(time);
60     auto modifier = this->modifier->getModifier();
61
62     for (auto &pair : this->items)
63         for (auto &item : pair.second)
64             item->setModifier(modifier);
65
66     for (; this->end > time
67           && this->requests > this->modifier->getPassed();
68           time += this->step)
69         for (auto &pair : this->items)
70             for (auto &item : pair.second)
71                 item->tick(time);
72 }

```

Листинг 3 – Реализация событийного подхода

```

1 class EventModel
2 {
3     public:
4         virtual ~EventModel(void) = default;
5         virtual void event(void) = 0;
6         virtual double nextEvent(void) const = 0;
7         virtual void generateNextEvent(void) = 0;
8         virtual void
9             setModifier(std::shared_ptr<RequestModifier>) = 0;
10 };
11
12 class EventRunner : public Runner
13 {
14     public:

```

```

15         EventRunner(size_t requests, double time,
16                     std::shared_ptr<EventRequestModifier>,
17                     std::list<std::shared_ptr<EventModel>>);
18     virtual ~EventRunner(void) override = default;
19
20     virtual void run(void) override;
21
22     private:
23         const size_t requests;
24         const double end;
25         std::shared_ptr<EventRequestModifier> modifier;
26         std::list<std::shared_ptr<EventModel>> items;
27 };
28
29 EventRunner::EventRunner(size_t requests, double time,
30
31     std::shared_ptr<EventRequestModifier>,
32
33     std::list<std::shared_ptr<EventModel>>)
34 : requests(requests), end(time), modifier(modifier),
35   items(items)
36 {
37     if (nullptr == this->modifier)
38         throw std::logic_error("Nullptr modifier");
39
40     for (auto &item : items)
41         if (nullptr == item)
42             throw std::logic_error("Nullptr modifier");
43 }
44
45 void EventRunner::run(void)
46 {
47     double time = 0;
48     this->modifier->setTime(time);
49     auto modifier = this->modifier->getModifier();
50
51     for (auto &item : this->items)
52         item->setModifier(modifier);
53
54     while (this->end > time
55           && this->requests > this->modifier->getPassed())

```

```

54     {
55         auto iter = this->items.begin(), next = iter;
56
57         for (; this->items.end() != iter; ++iter)
58             if ((*iter)->nextEvent() < (*next)->nextEvent())
59                 next = iter;
60
61         auto model = *next;
62         time = model->nextEvent();
63         model->event();
64         model->generateNextEvent();
65     }
66 }

```

Листинг 4 – Сборка схемы

```

1 void run_task(std::shared_ptr<RunnerBuilder> builder)
2 {
3     auto pipe1 = \
4     std::make_shared<BasicPipe>("pipe1");
5     auto pipe2 = \
6     std::make_shared<BasicPipe>("pipe2");
7     auto pipe3 = \
8     std::make_shared<BasicPipe>("pipe3");
9     auto pipe4 = \
10    std::make_shared<BasicPipe>("pipe4");
11    auto pipe5 = \
12    std::make_shared<BasicPipe>("pipe5");
13    auto pipe6 = \
14    std::make_shared<BasicPipe>("pipe6");
15
16    auto generator = \
17    std::make_shared<Generator>("generator", pipe1);
18    auto operator1 = \
19    std::make_shared<Processor>("operator1", pipe1, pipe2);
20    auto operator2 = \
21    std::make_shared<Processor>("operator2", pipe1, pipe2);
22    auto operator3 = \
23    std::make_shared<Processor>("operator3", pipe1, pipe3);
24    auto buffer1 = \
25    std::make_shared<Buffer>("buffer1", 0, pipe2, pipe4);
26    auto buffer2 = \

```

```

27     std::make_shared<Buffer>("buffer2", 0, pipe3, pipe5);
28     auto processor1 = \
29     std::make_shared<Processor>("processor1", pipe4, pipe6);
30     auto processor2 = \
31     std::make_shared<Processor>("processor2", pipe5, pipe6);
32
33     processor1->registerSender(buffer1);
34     processor2->registerSender(buffer2);
35
36     auto terminator1 = \
37     std::make_shared<Terminator>("terminator1", pipe1);
38     auto terminator2 = \
39     std::make_shared<Terminator>("terminator2", pipe6);
40
41     std::shared_ptr<StatisticsBlock> stats = \
42     std::make_shared<StatisticsBlock>("stats",
43         std::list<std::shared_ptr<Model>>{
44             pipe1, pipe2, pipe3, pipe4, pipe5, pipe6,
45             generator, operator1, operator2, operator3,
46             buffer1, buffer2,
47             processor1, processor2,
48             terminator1, terminator2
49         });
50
51     builder->addModel(generator)
52         .addModel(pipe1)
53         .addModel(operator1)
54         .addModel(operator2)
55         .addModel(operator3)
56         .addModel(terminator1)
57         .addModel(pipe2)
58         .addModel(pipe3)
59         .addModel(buffer1)
60         .addModel(buffer2)
61         .addModel(pipe4)
62         .addModel(pipe5)
63         .addModel(processor1)
64         .addModel(processor2)
65         .addModel(pipe6)
66         .addModel(terminator2)
67         .addModel(stats);

```

```

68
69     builder->create();
70     auto runner = builder->result();
71
72     runner->run();
73 }

```

Результаты работы

Генератор
Оператор
Обслуживающий аппарат

Минимум
8.000

Максимум
12.000

Моделирование

Метод
Дельта t

Число заявок
300

Время
10000.000

Минимальный шаг
0.100

Число замеров
10

Рассчитать

Результат

Время моделирования (среднее)	3048.92
Время пребывания в системе (среднее)	41.5724
Вероятность отказа (средняя)	0.210333
Вероятность отказа (максимальная)	0.22
Число отказов (среднее)	63.1

Рисунок 2 – Результат работы программы при заданных величинах

Вывод

В результате выполнения работы, было разработано программное обеспечение позволяющее определить вероятность для заданной схемы с произвольными параметрами установленных законов распределения.

Вероятность отказа для изначальных условий составила 0.21.