|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки 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 Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

**ОТЧЕТ О ВЫПОЛНЕНИИ ДОМАШНЕГО ЗАДАНИЯ ПО КУРСУ«АНАЛИТИЧЕСКИЕ МОДЕЛИ И ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕНА СИСТЕМНОМ УРОВНЕ»**

Студент Никатов Владислав Алексеевич

Группа РК6-84б

Тип задания Домашнее задание

Вариант 7

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ Никатов В.А. \_**

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ Берчун Ю.В. \_**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2020 г.*

**Оглавление**

[1 Задание 3](#_Toc41959069)

[2 Решение 3](#_Toc41959070)

[Приложение 1 7](#_Toc41959071)

# 1 Задание

Для многоканальной системы с неограниченной очередью, описанной в предыдущей лабораторной работе, построить таблицу отражающую состояния цепи текущих событий (ЦТС) и цепи будущих событий (ЦБС) по модельному времени. Решение задачи должно быть представлено в виде программного кода с использованием любого языка программирования, по усмотрению студента. В программе требуется реализовать алгоритм дискретно-событийного моделирования.

# 2 Решение

При решении задачи был реализован объект типа Transaction, хранящий в себе следующие значения:

* номер транзакта;
* время движения;
* номер текущего блока;
* уровень приоритета;
* номер следующего блока.

Под блоками понимаются объекты, которые в определенный момент времени содержат в себе транзакты, а так же выполняют над ними какие-либо действия. К объектам данного типа относятся:

1. **Generate.** При прохождении транзактом данного объекта, при входе транзакт обновляет значения текущего и следующего блоков на блок Generate и следующий за ним блок соответственно, а так же время движения на КМР (Как можно раньше). При выходе транзакта из блока создается новый транзакт в ЦБС, который попадет в блок через заданное функцией-генератором количество времени. Используется для генерации заявок на обслуживание, а так же транзакта, завершающего моделирование. В первом случае функция-генератор возвращает число, заданное экспоненциальным распределением со средним значением 39, во втором – время моделирования.
2. **Queue.** Объект может вмещать в себя ограниченное или неограниченное число объектов. При входе транзакта в очередь значение счетчиков объектов внутри увеличивается, при выходе – уменьшается.
3. **Storage.** Объект, исполняющий задачу многоканального обслуживания. Может иметь ограничение на число каналов. Транзакт может войти в объект только если счетчик внутри объекта меньше максимального заданного размера. При входе транзакта в объект значение счетчика увеличивается на 1, а сам транзакт перемещается в ЦБС, будущим событием является выход из объекта Storage, а время, через которое это произойдет, задается генератором, возвращающим числа, заданные экспоненциальным распределением со средним значением 229. Через указанный промежуток времени, транзакт вернется в ЦТС и будет готов выйти из объекта Storage. При выходе, счетчик объектов уменьшается.
4. **Terminate.** При входе транзакта в объект, значение счетчика модели уменьшается на заданное заранее число, а сам транзакт удаляется из модели.

Основным объектом является Model, в котором используются экземпляры перечисленных выше классов, хранятся ЦТС, ЦБС и ТМБ, а так же реализованы фазы ввода, коррекции таймера и просмотра. Используемые для моделирования объекты представлены и пронумерованы в порядке их вызова в таблице 2.1.

|  |  |
| --- | --- |
| № | Объект |
| 0 | Generate |
| 1 | Queue |
| 2 | Storage |
| 3 | Terminate |
| 4 | Generate |
| 5 | Terminate |

Состояния ЦТС и ЦБС сразу после стадии ввода, коррекции таймера и просмотра, а так же значение таймера модельного времени (ТМВ) отражены в таблице 2.2.2.

Таблица 2.2.2 а. Состояния ТМВ, ЦТС и ЦБС после каждой стадии.

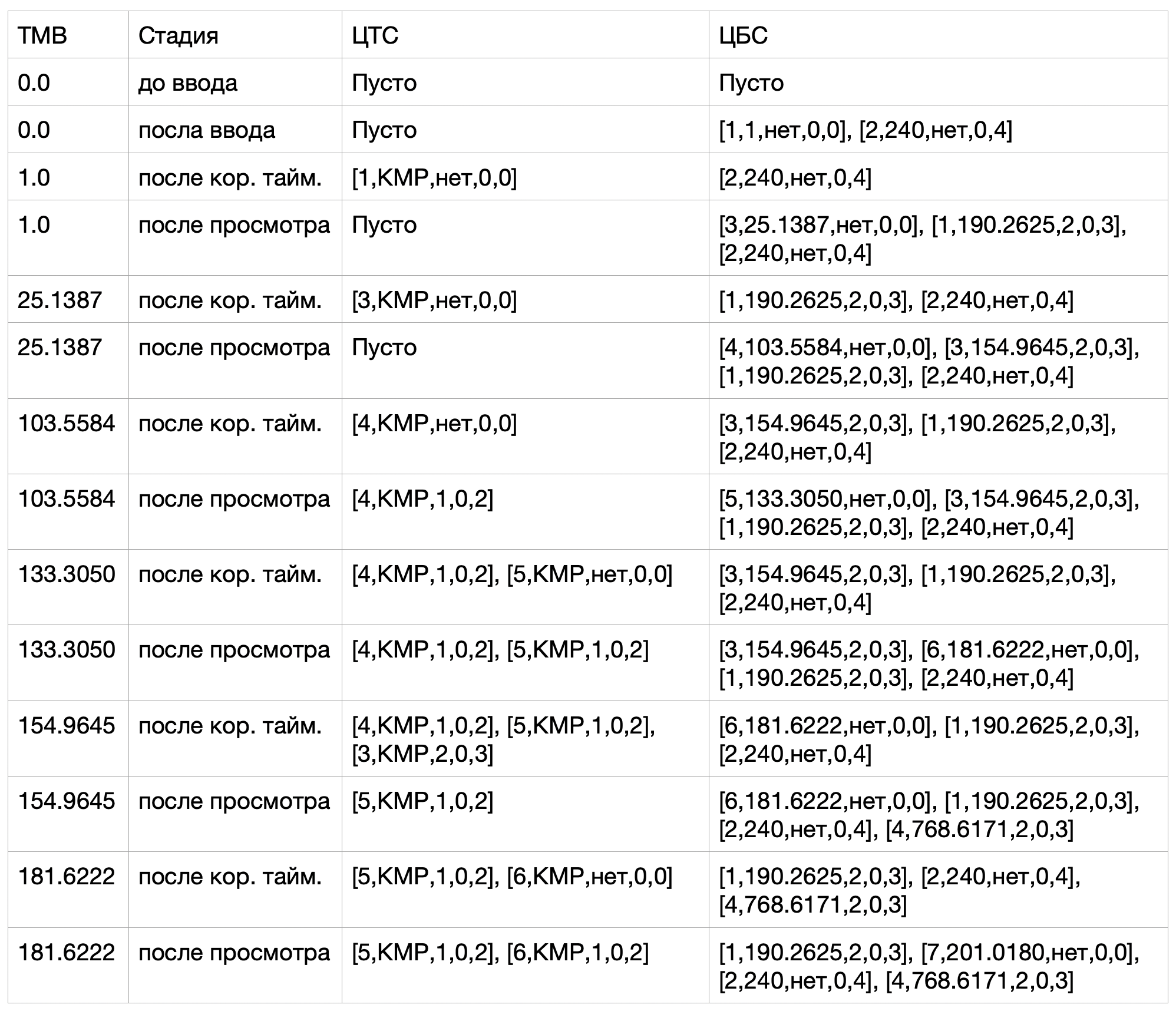
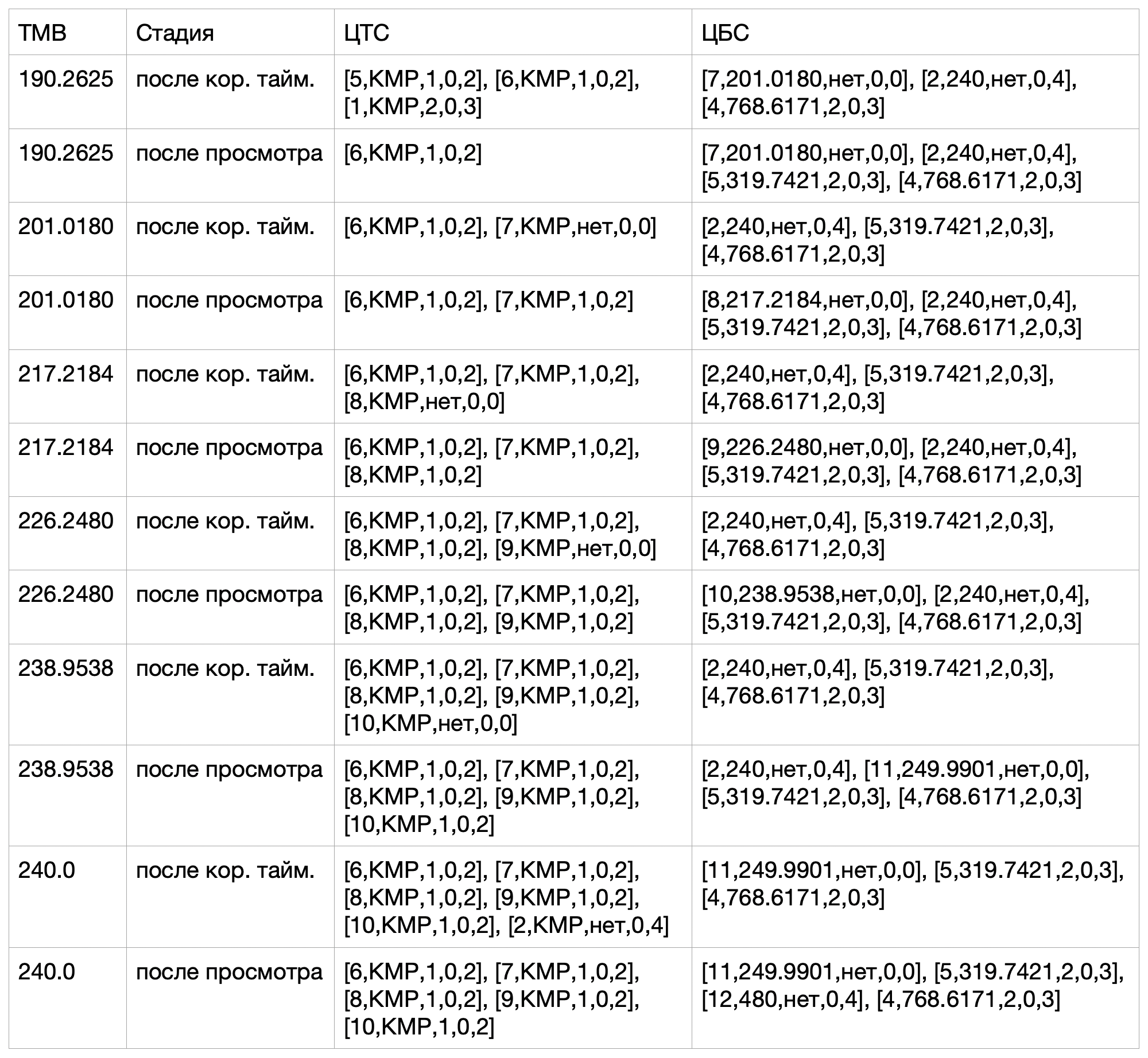


Таблица 2.2.2 б. Состояния ТМВ, ЦТС и ЦБС после каждой стадии.



# Приложение 1

**main.py**

**from** Model **import** Model  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 model = Model()  
 model.init()  
 model.run()

**Model.py**

**from** Generate **import** Generate  
**from** Terminate **import** Terminate  
**from** Queue **import** Queue  
**from** Storage **import** Storage  
**import** random  
  
  
**def** time\_generator(avg, first\_one=**False**):  
 **if** first\_one:  
 **yield** 1  
 **while True**:  
 **yield** random.expovariate(1./avg)  
  
  
**def** const\_generator(number):  
 **while True**:  
 **yield** number  
  
  
**class** Model:  
 **def** \_\_init\_\_(self):  
 self.\_cec = [] *# Сurrent Events Chain (цепь текущих событий)* self.\_fec = [] *# Future Events Chain (цепь будущих событий)* self.\_mtc = [0] *# Model Time Counter (счетчик времени моделирования)* self.\_count = [1]  
 self.\_blocks = []  
  
 **def** init(self):  
 self.\_blocks.append(Generate(fec=self.\_fec,  
 mtc=self.\_mtc,  
 time\_generator=time\_generator(39, first\_one=**True**)))  
 self.\_blocks.append(Queue())  
 self.\_blocks.append(Storage(cec=self.\_cec,  
 fec=self.\_fec,  
 mtc=self.\_mtc,  
 time\_generator=time\_generator(229),  
 max\_len=2))  
 self.\_blocks.append(Terminate(cec=self.\_cec,  
 counter=self.\_count))  
 self.\_blocks.append(Generate(fec=self.\_fec,  
 mtc=self.\_mtc,  
 time\_generator=const\_generator(240)))  
 self.\_blocks.append(Terminate(cec=self.\_cec,  
 counter=self.\_count,  
 decrement=1))  
  
 **def** run(self):  
 self.print\_info(**'До стадии ввода'**)  
 self.\_insert()  
 self.print\_info(**'После стадии ввода'**)  
 **while** self.\_count[0] != 0:  
 self.\_timer\_correction()  
 self.print\_info(**'После стадии коррекции таймера'**)  
 self.\_view()  
 self.print\_info(**'После стадии просмотра'**)  
  
 **def** \_insert(self):  
 **for** i, block **in** enumerate(self.\_blocks):  
 **if** isinstance(block, Generate):  
 block.generate(cur\_block=i)  
  
 **def** \_timer\_correction(self):  
 **if** len(self.\_fec) == 0:  
 exit(**'Произошла ошибка. Цепь будущих событий опустела раньше окончания программы.'**)  
 t = self.\_fec[0]  
 self.\_mtc[0] = t.transition\_time  
 self.\_fec.remove(t)  
 t.transition\_time = -1  
 self.\_cec.append(t)  
 **while** len(self.\_fec) != 0 **and** self.\_fec[0].transition\_time == self.\_mtc[0]:  
 t = self.\_fec[0]  
 self.\_fec.remove(t)  
 t.transition\_time = -1  
 self.\_cec.append(t)  
  
 **def** \_view(self):  
 f\_change = **True  
 while** f\_change:  
 f\_change = **False  
 for** t **in** self.\_cec:  
 **while** self.\_blocks[t.current\_block].can\_exit() **and** self.\_blocks[t.next\_block].can\_enter():  
 f\_change = **True** *# print('Переход транзакта', t.num, 'из блока', t.current\_block, 'в блок', t.next\_block)* **if** t.current\_block != -1:  
 self.\_blocks[t.current\_block].exit()  
 self.\_blocks[t.next\_block].enter(t)  
 **if** len(self.\_cec) == 0:  
 **return** t = self.\_cec[0]  
  
 **def** print\_info(self, stage: str):  
 print(stage + **':'**)  
 print(**'ТМВ:'**, self.\_mtc[0])  
 print(**'ЦТС:'**, self.\_cec)  
 print(**'ЦБС:'**, self.\_fec)  
 print()

**Generate.py**

**from** Transaction **import** Transaction  
**from** operator **import** attrgetter  
  
  
**class** Generate:  
 \_counter = 1  
  
 **def** \_\_init\_\_(self,  
 fec: list,  
 mtc: list,  
 time\_generator,  
 priority: int = 0):  
 self.\_fec = fec  
 self.\_mtc = mtc  
 self.\_priority = priority  
 self.\_time\_generator = time\_generator  
 self.cur\_transaction = **None  
  
  
 def** generate(self, cur\_block):  
 self.\_fec.append(Transaction(num=Generate.\_counter,  
 transition\_time=self.\_mtc[0] + next(self.\_time\_generator),  
 current\_block=-1,  
 priority=self.\_priority,  
 next\_block=cur\_block))  
 Generate.\_counter += 1  
 self.\_fec.sort(key=attrgetter(**'transition\_time'**))  
  
 **def** can\_enter(self):  
 **return True  
  
 def** can\_exit(self):  
 **return True  
  
 def** enter(self, t):  
 **if** self.cur\_transaction **is not None**:  
 print(**'Потеря транзакта'**, t)  
 **return** t.current\_block = t.next\_block  
 t.next\_block = t.next\_block + 1  
 t.transition\_time = -1  
 self.cur\_transaction = t  
  
 **def** exit(self):  
 t = self.cur\_transaction  
 self.generate(t.current\_block)  
 self.cur\_transaction = **None  
 return** t

**Storage.py**

**from** operator **import** attrgetter  
  
  
**class** Storage:  
 **def** \_\_init\_\_(self,  
 cec: list,  
 fec: list,  
 mtc: list,  
 time\_generator,  
 max\_len: int = **None**):  
 self.\_cec = cec  
 self.\_fec = fec  
 self.\_mtc = mtc  
 self.\_time\_generator = time\_generator  
 self.\_m\_len = max\_len  
 self.\_s = 0  
  
 **def** can\_enter(self):  
 **if** self.\_m\_len **is None**:  
 **return True  
 return** self.\_s < self.\_m\_len  
  
 **def** can\_exit(self):  
 **return True  
  
 def** enter(self, t):  
 self.\_s += 1  
 t.current\_block = t.next\_block  
 t.next\_block = t.next\_block + 1  
 t.transition\_time = self.\_mtc[0] + next(self.\_time\_generator)  
 self.\_fec.append(t)  
 self.\_cec.remove(t)  
 self.\_fec.sort(key=attrgetter(**'transition\_time'**))  
  
 **def** exit(self):  
 self.\_s -= 1

**Queue.py**

**from** collections **import** deque  
  
  
**class** Queue:  
 **def** \_\_init\_\_(self, max\_len: int = **None**):  
 self.\_m\_len = max\_len  
 self.\_q = deque()  
  
 **def** can\_enter(self):  
 **if** self.\_m\_len **is None**:  
 **return True  
 return** len(self.\_q) < self.\_m\_len  
  
 **def** can\_exit(self):  
 **return True  
  
 def** enter(self, t):  
 t.current\_block = t.next\_block  
 t.next\_block = t.next\_block + 1  
 t.transition\_time = -1  
 self.\_q.appendleft(t)  
  
 **def** exit(self):  
 **return** self.\_q.pop()

**Terminate.py**

**class** Terminate:  
 **def** \_\_init\_\_(self,  
 counter: list,  
 cec: list,  
 decrement: int = 0):  
 self.\_cec = cec  
 self.\_counter = counter  
 self.\_decrement = decrement  
  
 **def** can\_enter(self):  
 **return True  
  
 def** can\_exit(self):  
 **return True  
  
 def** enter(self, t):  
 self.\_cec.remove(t)  
 self.\_counter[0] -= self.\_decrement  
  
 **def** exit(self, t):  
 **pass**

**Transaction.py**

**class** Transaction:  
 **def** \_\_init\_\_(self,  
 num: int,  
 transition\_time: float,  
 current\_block: int,  
 priority: int,  
 next\_block: int):  
 self.num = num  
 self.transition\_time = transition\_time  
 self.current\_block = current\_block  
 self.priority = priority  
 self.next\_block = next\_block  
  
 **def** \_\_str\_\_(self):  
 string = **'['** + str(self.num) + **','  
 if** self.transition\_time == -1:  
 string += **'КМР'  
 else**:  
 string += str(self.transition\_time)  
 string += **','  
 if** self.current\_block == -1:  
 string += **'нет'  
 else**:  
 string += str(self.current\_block)  
  
 string += **','** + str(self.priority) + **','** + str(self.next\_block) + **']'  
 return** string  
  
 **def** \_\_repr\_\_(self):  
 **return** str(self)