Практическая работа №5

Тема: «Стэк и очередь».

Цель работы: изучить СД «стэк» и «очередь» научиться их программно реализовывать.

Реализовать систему, представленную на рисунке 1.

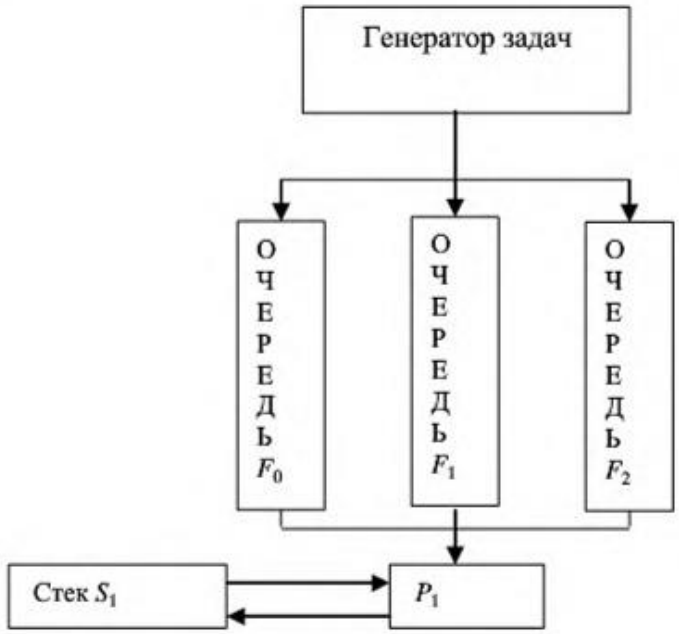

Рисунок 1 – Система для реализации

Задачи из очередей берутся по очереди. Задачи из первой и второй очереди выполняются либо на первом процессоре, либо на втором процессоре, если оба свободны, то на первом. Задачи из очереди третьей выполняются на первом и втором, если оба свободны, то на втором.

Реализуем генератор задач, который будет состоять из структуры и класса, который предоставляет доступ к полям (Рисунок 2).

```python
from dataclasses import dataclass

from numpy import random as rnd


@dataclass()
class TaskData:
    time: int = None
    task_type: int = None

class Task():
    def __init__(self):
        time_work = [3, 6, 9]
        task_type = rnd.randint(high=3, low=0)
        self.current_task = TaskData()
        self.current_task.time = time_work[task_type]
        self.current_task.task_type = task_type


    def get_time(self):
        return self.current_task.time

    def get_type(self):
        return self.current_task.task_type

    def set_time(self, time):
        self.time = time

    def set_type(self, type)
        self.task_type = type
```

Рисунок 2 – Генератор задач.

Реализуем процессор, у которого будет два потока, которые представим структурой (Рисунок 4). Диаграмма деятельности для добавления задачи на выполнение представлена на Рисунке 3.

Рисунок 3 - Диаграмма деятельности для добавления задачи.

```
from dataclasses import dataclass
from task import Task

@dataclass()
class Thread:
    time_work: int = None
    task_type: int = None
    idle: bool = True

class Processor():
    def __init__(self):
        self.p = Thread()

    def add_task(self, task: Task):
        if self.p.task_type < task.get_type():
            l = Task()
            l.set_type(task.get_type())
```

```
            l.set_time(task.get_time())
            stack.add_item(l)
            self.p.time_work = task.get_time()
            self.p.task_type = task.get_type()
        elif self.idle_proc():
            self.p.time_work = task.get_time()
            self.p.task_type = task.get_type()
        else:
            stack.add_item(task)

    def __task_perform_p(self):
        self.p.time_work -= 1
        if self.p.time_work <= 0:
            self.p.idle = True
            self.p_task_type = None

    def __str__(self):
        string = "|proc|type|time|idle|"
        if not self.p.idle:
            string += "\n|1   |{:<4}|{:<4}|{:<4}|".format(str(self.p.task_type),
str(self.p.time_work), str(self.p.idle))
        else:
            string += "\n|1   |None|None|True|"
        return string

    def work(self):
        if not self.p.idle:
            self.__task_perform_p()
        else:
            self.p.idle = True

    def idle_proc(self):
        return self.p.idle
```

Рисунок 4 - Класс процессора.

Реализуем класс очереди, диаграммы деятельности для добавления задачи в очереди и ее удаления из очереди представлены на Рисунках 5 и 6 соответственно, листинг класса представлен на Рисунке 7.

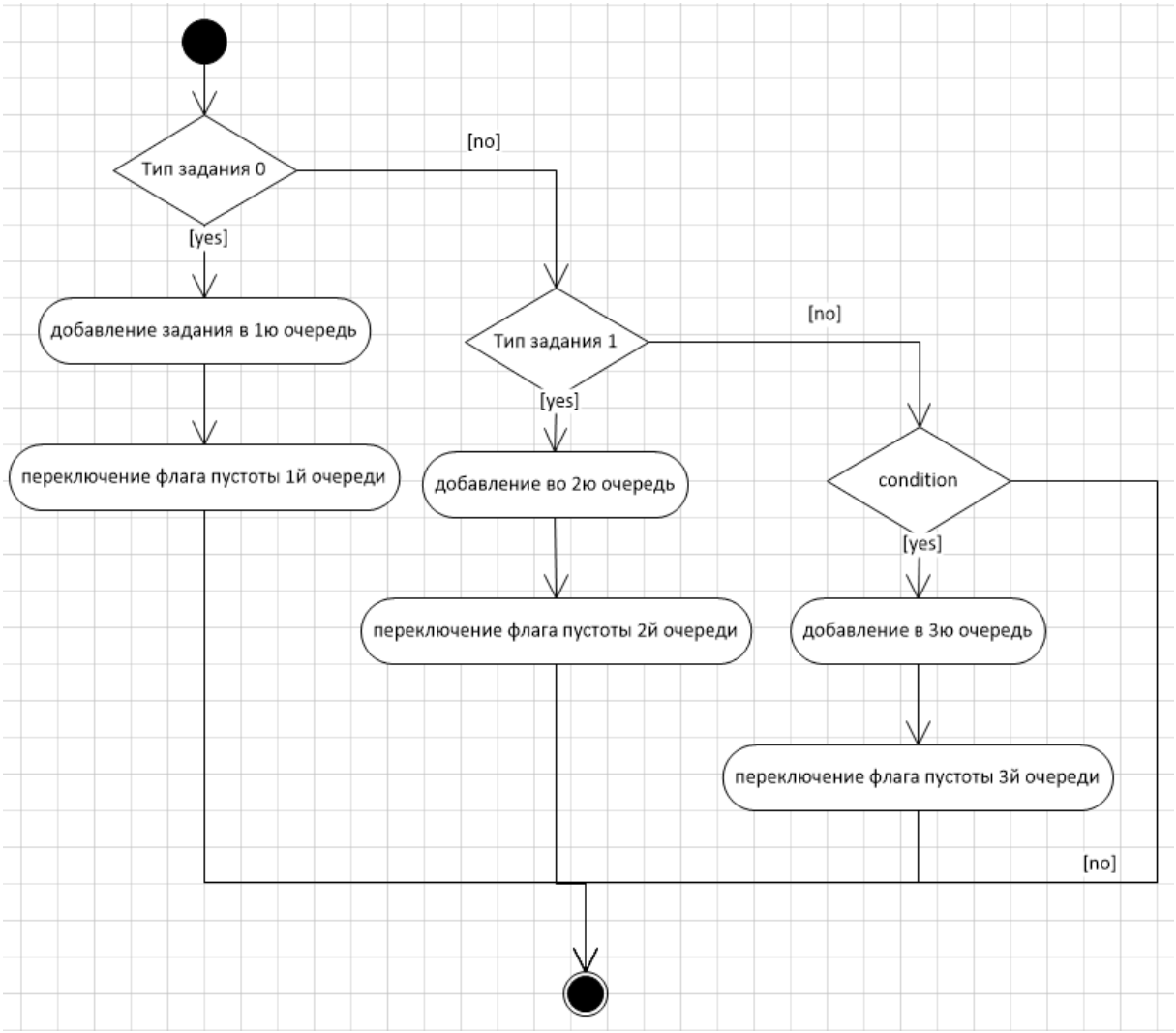

Рисунок 5 – Добавление задачи в очередь.

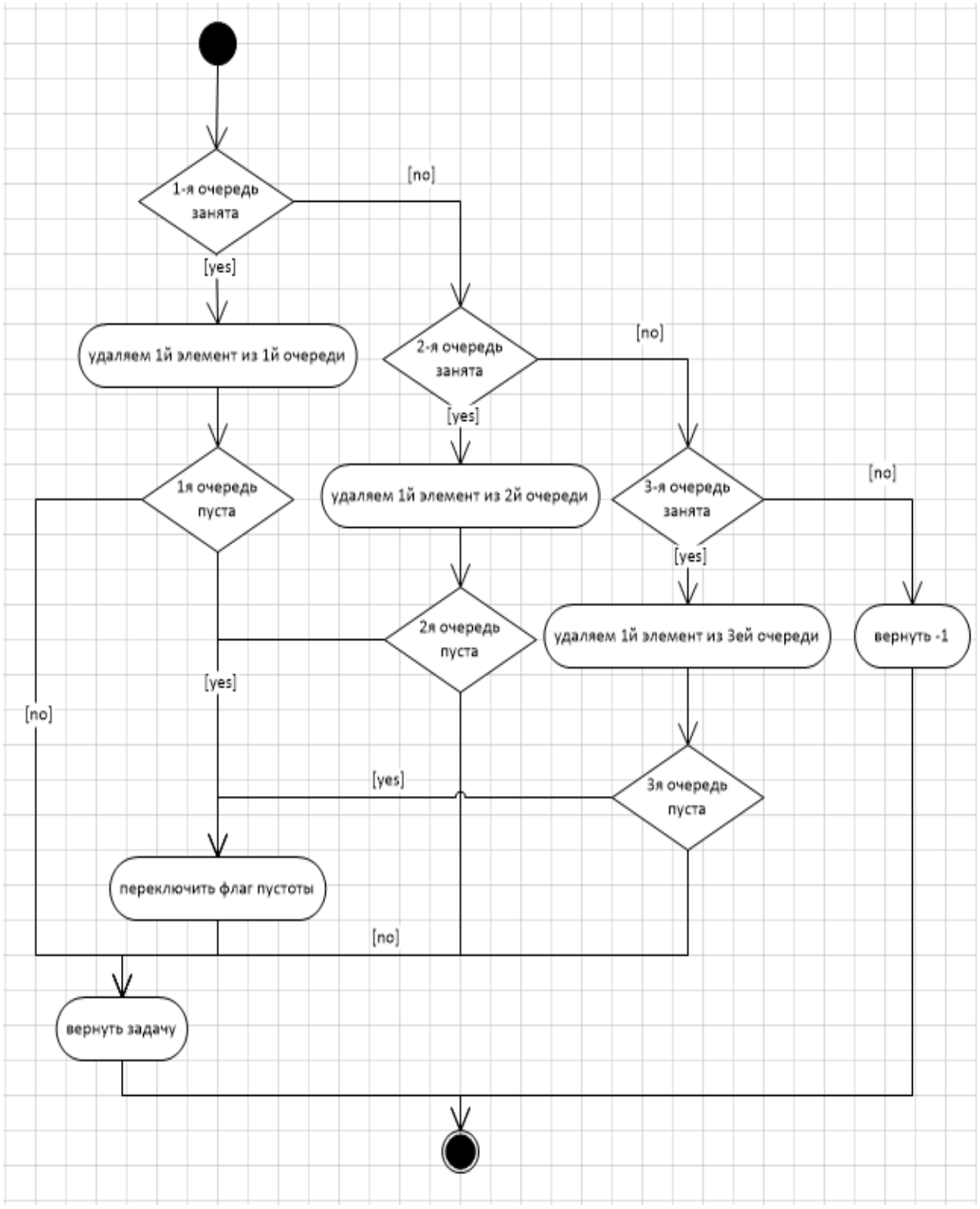

Рисунок 6 – Удаление элемента из очереди.

```
From dataclasses import dataclass
from task import Task


@dataclass()
class QueueData:
    task_type = []
    is_empty: bool = True

class Queue():

    def __init__(self):
        self.q1 = QueueData()
        self.q2 = QueueData()
        self.q3 = QueueData()

    def add_task(self, task:Task):
        if task.get_type() == 0:
            self.q1.task_type.append(task)
            self.q1.is_empty = False
        elif task.get_type() == 1:
            self.q2.task_type.append(task)
            self.q2.is_empty = False
        elif task.get_type() == 2:
            self.q3.task_type.append(task)
            self.q3.is_empty = False

    def del_task(self):
        if not self.q1.is_empty:
            task = self.q1.task_type.pop(0)
            if len(self.q1.task_type) == 0:
                self.q1.is_empty = True
        elif not self.q2.is_empty:
            task = self.q2.task_type.pop(0)
            if len(self.q2.task_type) == 0:
                self.q2.is_empty = True
        elif not self.q3.is_empty:
            task = self.q3.task_type.pop(0)
            if len(self.q3.task_type) == 0:
                self.q3.is_empty = True
        else:
            task = -1
        return task

    def __str__(self):
        return str(str(self.q1.task_type) + str(self.q1.is_empty) + str(self.q2.task_type)
+ str(self.q2.is_empty) + str(self.q3.task_type) + str(self.q3.is_empty))

    def get_queue_empty_flag(self):
        return self.q1.is_empty and self.q2.is_empty and self.q3.is_empty
```

Рисунок 7 - Очередь задач.

Реализуем стэк задач, диаграмма деятельности для добавления в стэк и удаления задачи из стека представлена на рисунках 7 и 8 соответственно. Листинг реализации представлен на Рисунке 9.

Рисунок 7 - Добавление задачи в стэк.

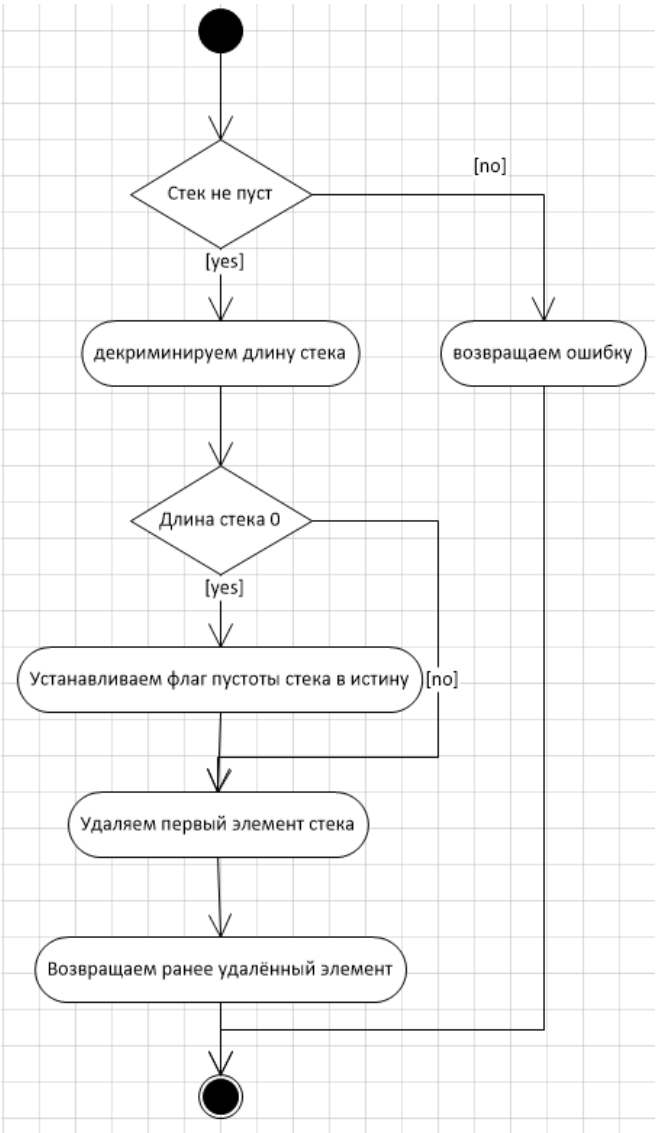

Рисунок 8 - Удаление элемента из стека.

```
from  dataclasses import dataclass

@dataclass()
class TaskStack:
    task_list = []
    is_empty = True
    length = 0

class Stack:
    def __init__(self):
        self.stack = TaskStack

    def add_item(self, task):
        self.stack.task_list.append(task)
        self.stack.length += 1
        self.stack.is_empty = False

    def del_item(self):
        if self.stack.length != 0:
            self.stack.length -= 1
            if self.stack.length == 0:
                self.stack.is_empty = True
            return self.stack.task_list.pop(len(self.stack.task_list)-1)
        return -1

    def check_is_empty(self):
        return self.stack.is_empty

    def get_length(self):
        return self.stack.length

    def __str__(self):
        strok = "|type|time|"
        if not self.stack.is_empty:
            for task in self.stack.task_list:
                strok += "\n|{:<4}|{:<4}|".format(str(task.get_type()),
str(task.get_time()))
        else:
            strok += "\n|None|None|"
        strok += "\n|____|____|\n\n"
        return strok
```

Рисунок 9 - Реализация стека задач.

Реализуем основную логику программы, диаграмма деятельности для которой представлена на рисунке 10. Рисунок 11 содержит код файла main.py.

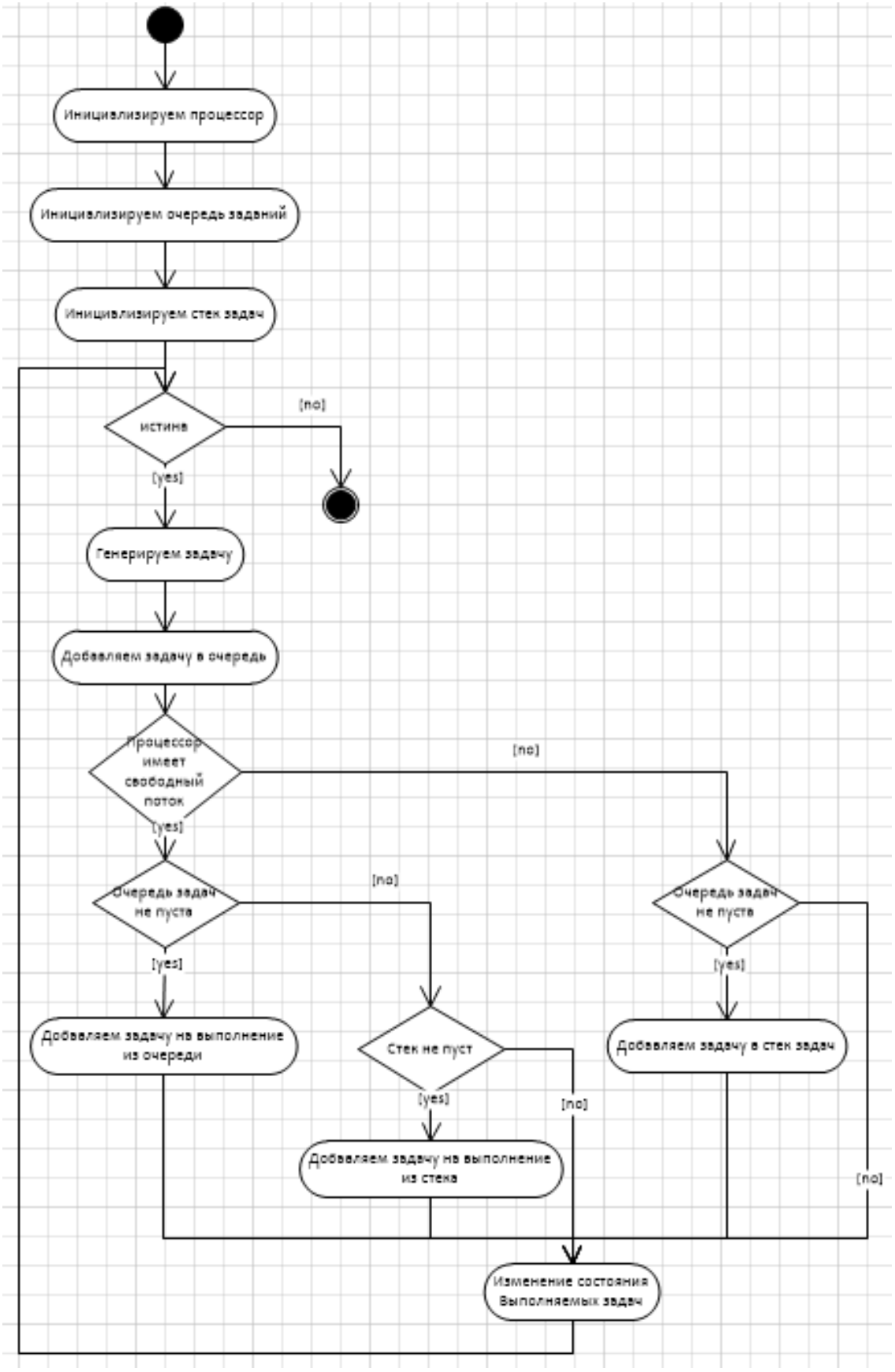

Рисунок 10 - Диаграмма деятельности для главной логики программы.

```
from processor import Processor
from queue import Queue
from task import Task
from stack import Stack

if __name__ == "__main__":
    proc = Processor()
    task_queue = Queue()
    task_stack = Stack()
    while True:
        a = Task()
        task_queue.add_task(a)
        if proc.idle_proc():
            if not task_queue.get_queue_empty_flag():
                proc.add_task(task_queue.del_task())
            elif not task_stack.check_is_empty():
                proc.add_task(task_stack.del_item())
        else:
            if not task_queue.get_queue_empty_flag():
                task_stack.add_item(task_queue.del_task())
        print(proc)
        print(task_stack)
        print(task_queue)
        proc.work()
```

Рисунок 11 - Файл main.py.

Вывод: в ходе работы были изучены структуры данных стек и очередь.