**Практическая работа №5**

**Тема: «Стэк и очередь».**

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

Реализовать систему, представленную на рисунке 1.

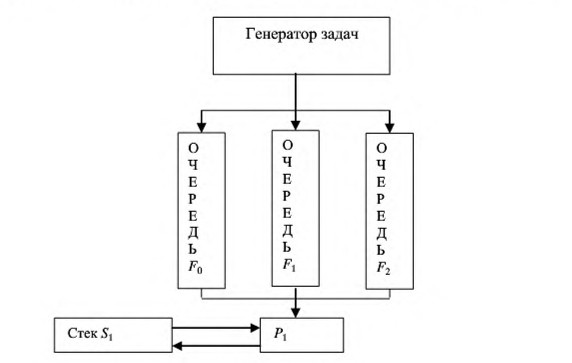


Рисунок 1 – Система для реализации

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).

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.

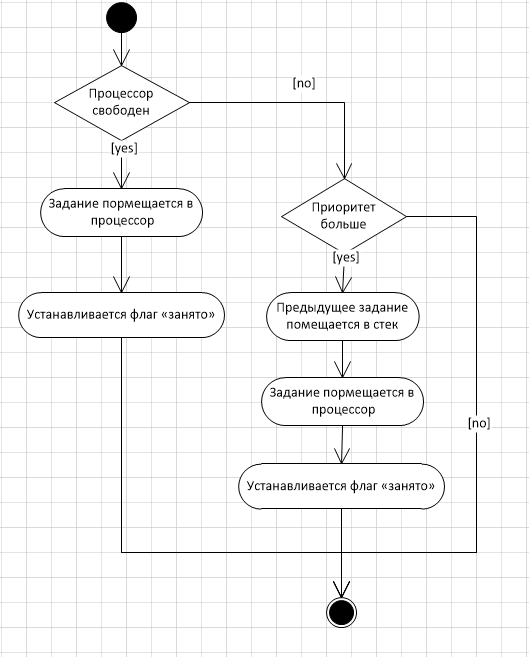


Рисунок 3 - Диаграмма деятельности для добавления задачи.

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.

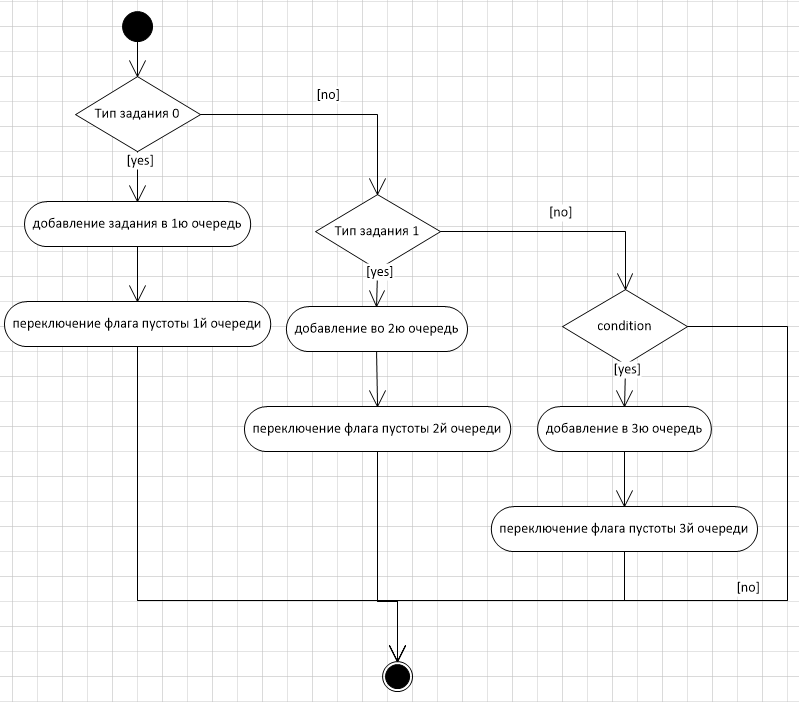


Рисунок 5 – Добавление задачи в очередь.

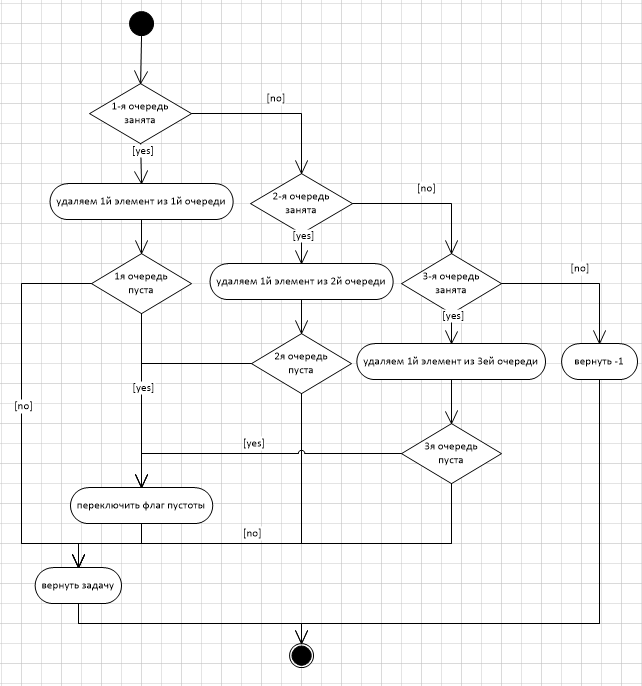


Рисунок 6 – Удаление элемента из очереди.

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.

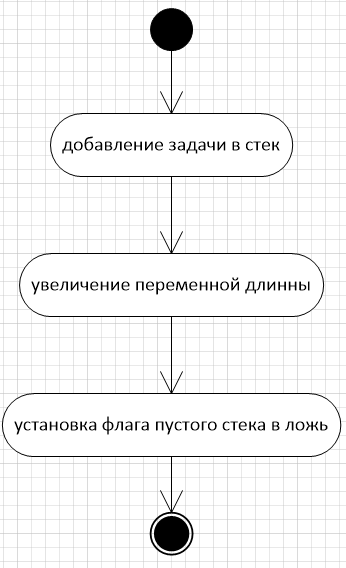


Рисунок 7 - Добавление задачи в стэк.

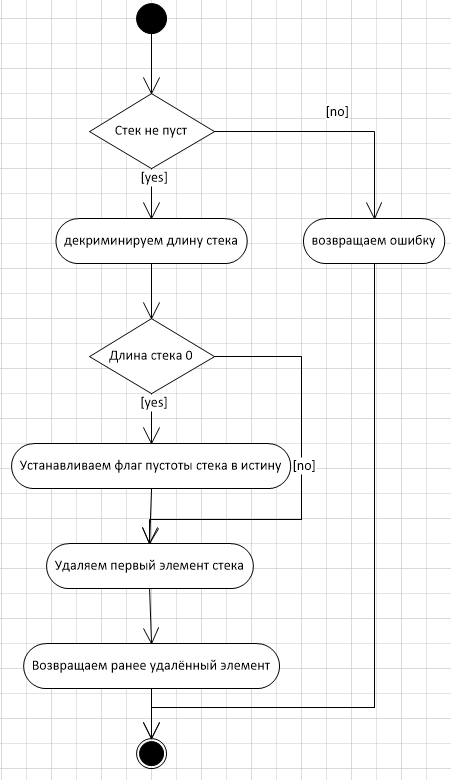


Рисунок 8 - Удаление элемента из стека.

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.

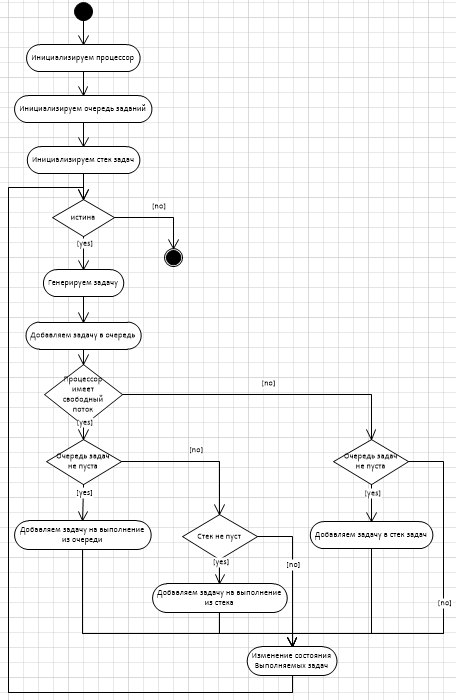


Рисунок 10 - Диаграмма деятельности для главной логики программы.

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.